Publication Number 0117-0369-10 Rev.A

October 1984

USER DEFINABLE DISASSEMBLER, UDD-205

USER'S MANUAL

Gould Inc., Design & Test Systems Division
4650 Old Ironsides Drive
Santa Clara, CA 95054-1279
Telephone: (408) 988-6800
TWX/TELEX # 910-338-0509

Copyright © 1984. No part of this publication may be reproduced without written permission from Gould, Inc., Design and Test Systems Division. Printed in U.S.A.

10/84

ERRATUM

Due to an error in release 1.2 of the K205 software, it is necessary always to specify at least one ADDRESS group section using the TGP database instruction. This is contrary to the TGP instruction that states that if no address group is specified, samples are assumed to be addressed in sequential order. Release 1.3, and above, of the K205 software will correct this problem.

PREFACE

The Gould UDD-205 User Defined Disassembler is an accessory of the K205 Logic Analyzer. A disassembler provides the user with a means of altering the data display of the K205 to suit a specific application. The UDD-205 is designed for use with the K205 with a microprocessor not currently supported by the Gould Disassembler product line. The UDD-205 may also substitute for a specific disassembler if such disassembler is not immediately available.

As a result of its "universal" characteristics, there may be instances where the UDD-205 may not properly handle the target device.

This manual presents a brief overview of microprocessor architecture, basic theory of internal operation of the UDD-205, and complete instructions for programming and operating the UDD-205. The operating instructions are separated into three chapters reflecting the three parts of the UDD-205: Editor, Preprocessor, and Disassembler. Appendices illustrate the worksheets and final source file used to generate a UDD-205 database for the Intel 8085 microprocessor.

The user should have a basic understanding of microprocessors and computer architecture. Where examples are required to illustrate UDD-205 theory or operation, reference to the Intel 8085 microprocessor is used throughout this manual.

The material in this manual reflects the software valid on June 15, 1984, and is up-to-date at the time of publication, but is subject to change without notice.

RELATED PUBLICATIONS

Other manuals associated with these products include the K205 Logic Analyzer Users Manual, and the operating manuals related to any components used in a system involving the Logic Analyzers.

ASSISTANCE

If you require assistance with this product, please contact Gould Inc., Design and Test Systems Division, at the address on the title page of this manual.

NOTICE

The Gould Inc., Design and Test Systems Division, makes no warranty of any kind regarding the User Definable Disassembler, UDD-205, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose. Gould Inc., Design and Test Systems Division, assumes no responsibility for errors that may appear in this manual, and makes no commitment to update the information contained in this manual.

Copies of this manual and other Gould Inc., Design and Test Systems Division, literature can be obtained from:

Gould Inc., Design and Test Systems Division
4560 Old Ironsides Drive
Santa Clara, CA 95054-1279

CONTENTS

Chapter		Page
1	INTRODUCTION	1-1
	THE UDD-205 SYSTEM	1-1
	USING THIS MANUAL	1-2
	PROGRAMMING	1-2
	INSTALLING THE UDD-205	1-3
	USING THE UDD-205	1-3
	UTILITIES	1-4
2	GETTING STARTED	2-1
	GENERAL	2-1
	CREATE DATABASE SOURCE FILE	2-2
	References	2-2
	Preparation	2-2
	Prepare Source-File Entries	2-2
	Install Terminal for Data Entry	2-4
	Make Source-File Entries	2-5
	GENERATE DATABASE	2-6
	References	2-6
	Operations	2-6
	OPERATE THE UDD-205	2-6
	References	2-6
	Operations	2-6
3	THEORY OF OPERATION	3-1
	GENERAL	3-1
	MICROPROCESSOR INSTRUCTION ARCHITECTURE	3-1
	Instruction Opcode Description	3-1
	Example of a Simple Opcode	3-1
	Example of a Complex Opcode	3-2
	OVERVIEW OF THE UDD	3-3
	DATA ALIGNMENT	3-6
	Create Trace Arrays	3-6
	Mark Known First-Instruction Bytes	3-6
	Test Validity and Size of Samples	3-7
	Create Trace_Index Array	3-8
	Primitive Identification of Other Status States	3-8
	DISASSEMBLY DISPLAY	3-9
	Characteristics of a Disassembly Display Line	3-8
	Select Samples for Display	3-9
	INSTRUCTION DECODE	3-9
	Characteristics of an OPC Specifier	3-9
	Source Field Generation	3-10
	Determine if Decode is Complete or Continues Branching	3-11
	Branching	3-11
	Translating	3-12
	Build an OPC Decode Tree	3-14

4 PROGRAMMING. 4-1
GENERAL 4-1
TRACE GROUPS AND STATUS INFORMATION 4-1
Trace_Group Specifier. 4-1
Signal Grouping. 4-2
Status Information 4-2
MACHINE_BLOCK (MCH) 4-3
Display Lines. 4-3
GDP Specifiers 4-4
No-Data Situation. 4-5
VALID TRACE CODES 4-6
Identify Starting Display-line cycles. . . . 4-6
Determine Format of Display Line 4-6
DECODING AND DISPLAYING MICROPROCESSOR
INSTRUCTION SET 4-7
First Decode Method 4-9
Second Decode Method 4-10
Third Decode Method. 4-11
INSTRUCTION BLOCK (INS) 4-12

5 INSTRUCTION SET. 5-1
GENERAL 5-1
Statement Identifier 5-1
Description. 5-1
Format 5-1
Default. 5-1
INPUT FILE CONSTRUCTION 5-1
INSTRUCTION INDEX 5-3
TRACE DATA FORMAT SPECIFIERS. 5-5
TGP. 5-6
CHANNEL FORMAT SPECIFIERS 5-7
THR. 5-8
VAR. 5-9
POL. 5-10
CLOCK SPECIFIERS. 5-11
CLK. 5-12
LABEL SPECIFIERS. 5-15
NAM. 5-16
HDG. 5-17
VER. 5-18
HLP. 5-19
STATUS SPECIFIERS 5-21
MSK. 5-22
CHK. 5-23
VTC. 5-24
DMD. 5-25
TCS. 5-26
MMD. 5-27
DECODE ARRAYS 5-29
OPC. 5-30
INS. 5-31
MCH. 5-32
GDP. 5-33
VLD. 5-35
OCL. 5-36
TPL. 5-37
GDL. 5-38

6 UDD UUTIL OPERATION. 6-1
GENERAL 6-1
Commands 6-1
Filename Entry 6-2
K206 Function Keys 6-2
UTILITY COMMANDS. 6-3
EDIT Command 6-3
KLIST Utility Command. 6-3
TLIST Utility Command. 6-4
LOCK and UNLOCK Utility Commands 6-5
SETTERM Utility Command. 6-5
DOWNLINK Utility Command 6-7
EXIT Utility Command 6-7
TERMINAL OPERATION. 6-8
UDD Editor 6-8
Editor Text Pointer. 6-9
Upper/Lower-Case Characters. 6-9
Line Editor. 6-9
TERMINAL EDIT COMMANDS. 6-10
Edit Commands. 6-10

7 UDD PREPROCESSOR OPERATION 7-1
GENERAL 7-1
Commands 7-1
Begin Operations 7-2
PREPROCESSOR INPUT-FILE CONSTRUCTION. 7-2
PREPROCESSOR OUTPUT FILES 7-2
Database-Object File 7-2
List File. 7-3
Output-File Controls 7-3
LIST-FILE INFORMATION 7-3
KLIST UTILITY COMMAND 7-4
TLIST UTILITY COMMAND 7-4
EXIT UTILITY COMMAND. 7-5

8 UDD DISASSEMBLER OPERATION 8-1
GENERAL 8-1
Connections. 8-1
Load the UDD 8-1
RECORD SETUP. 8-2
KEYBOARD OPERATION. 8-2
Display Operations 8-2
Display Format, Mnemonic/Object Format . . . 8-2
Display Format, Shift/S Set Mode 8-3
Disassembler Search Word 8-4
Trace-Control Pattern Selection. 8-4
DISASSEMBLER SCREEN DISPLAY 8-5
DISASSEMBLER HELP FACILITY. 8-6
DISASSEMBLER NOTES. 8-6
Software Version Numbers, Shift/V. 8-6
Processing Indications 8-7
SAMP Display 8-7

Appendices

		Page
A	PREPROCESSOR EXAMPLE - STATE MACHINE	A-1
B	PREPROCESSOR EXAMPLE - 8085 MICROPROCESSOR	B-1
C	SAMPLE PREPROCESSOR WORKSHEETS - 8085 MICROPROCESSOR	C-1
D	PREPROCESSOR WORKSHEETS	D-1
	User Definable Disassembler OC Worksheet	D-2
	User Definable Disassembler GD Worksheet	D-3
E	SAMPLE DISASSEMBLER FORMAT SCREEN	E-1
F	SAMPLE DISASSEMBLER CLOCK SCREEN	F-1
G	SAMPLE DISASSEMBLER TRACE CONTROL SCREEN	G-1
H	SAMPLE DISASSEMBLER DATA DISPLAY SCREEN	H-1
I	ERROR AND WARNING MESSAGES	I-1
J	RS-232 PORT SPECIFICATIONS	J-1
K	ABBREVIATIONS	K-1

Figures

		Page
1-1	Overview of the UDD-205 System	1-1
2-1	Functions of the UDD-205 System	2-1
3-1	UDD Structure	3-4
3-2	UDD Sample Processing	3-5
6-1	UDD Editor Files System	6-8
J-1	Null Modem	J-2

Tables

		Page
2-1	Source-File Entries	2-3
4-1	Summary of Decode Methods	4-8
5-1	Predefined Literals	5-2
5-2	Index to UDD Instructions	5-3
6-1	UUTIL Command-Line Quick Keys	6-1
6-2	UUTIL Command-Line Function Keys	6-3
6-3	KLIST Function Keys	6-4
6-4	Terminal Control Commands	6-4
6-5	Utility Command Codes	6-6
6-6	Line-Editor Control Characters	6-9
6-7	UDD Edit Commands	6-10
7-1	UDDGEN Preprocessing Commands	7-1
7-2	TLIST F-Key Functions	7-4
7-3	Terminal Control Commands	7-5
8-1	Disassembler Soft-Key Functions	8-3
8-2	Set-Mode Soft-Key Functions	8-3
8-3	CRT Display Lines	8-5
8-4	CRT Display Columns	8-6

INTRODUCTION

THE UDD-205 SYSTEM

The UDD-205 User Definable Disassembler system provides a means for altering the data display of the Logic Analyzer to suit a specific application. The UDD-205 decodes record samples obtained by the Logic Analyzer, and displays them in a format designed by the user.

The user designs a database for use by the UDD-205. The UDDGEN preprocessor utility changes the database to a format usable by the system. Other utilities, including an editor, down loading, and listing programs, facilitate use of the UDD system. An overview of the UDD-205 is shown schematically in Figure 1-1.

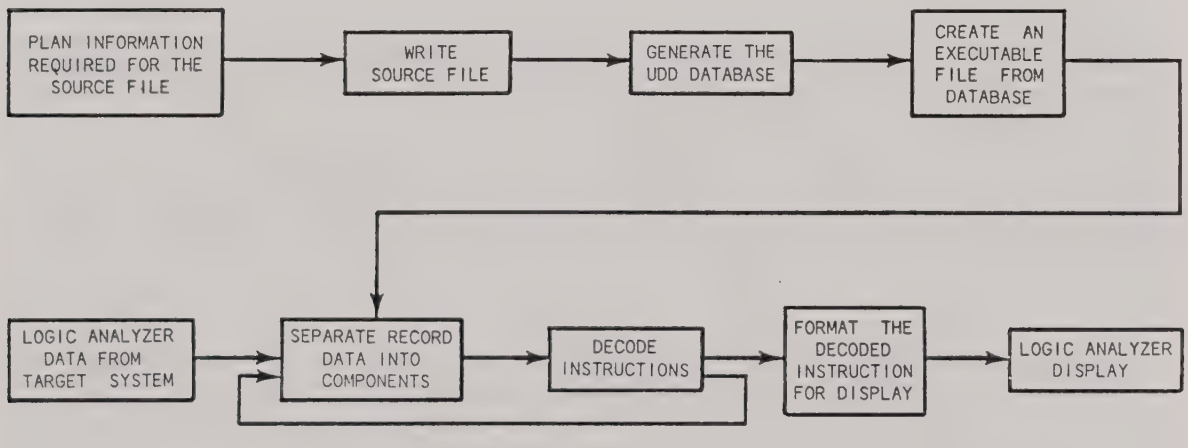

PLAN INFORMATION REQUIRED FOR THE SOURCE FILE
WRITE SOURCE FILE
GENERATE THE UDD DATABASE
CREATE AN EXECUTABLE FILE FROM DATABASE
LOGIC ANALYZER DATA FROM TARGET SYSTEM
SEPARATE RECORD DATA INTO COMPONENTS
DECODE INSTRUCTIONS
FORMAT THE DECODED INSTRUCTION FOR DISPLAY
LOGIC ANALYZER DISPLAY

Figure 1-1. Overview of the UDD-205 System

USING THIS MANUAL

This manual is intended to provide guidelines for programming the Gould User Definable Disassembler, UDD-205. It is assumed that the UDD programmer is thoroughly familiar with the target microprocessor and programming techniques. This manual also provides information for connecting and using the UDD-205.

Although this manual is organized according to a logical sequence, the user will find certain chapters more relevant than others. The following sections suggest chapters to refer to for the four aspects of UDD utilization:

- Programming the UDD
- Connecting with the target microprocessor
- Using the UDD
- Utilities

Appendices, supporting the various chapters, are referred to in the text where appropriate. Appendix D contains blank worksheets that can be reproduced for constructing the database that programs the UDD.

PROGRAMMING

In addition to having general knowledge about programming and the target microprocessor the programmer needs to know how to construct the UDD database, and the nature of the available parameters.

The UDD programmer may find Chapter 3, Theory of Operations, useful for understanding how the User Definable Disassembler operates. This understanding can enhance the programmer's ability to program the UDD, but does not provide instructions for programming.

Chapter 4, Programming, is more prescriptive. There is no unique procedure for programming the UDD; therefore no simple recipe can be given to guide the programmer. Programming is presented by way of the example of building a database for the 8085 microprocessor. By observing that example, the programmer can gain insight into techniques into building a database that suits his requirements.

Programming is accomplished by building a source file. The Source file is converted by the UDDGEN facility to a list file (.LST) and executable file (.EXE), the active database. Two worksheets, located in Appendix D, are provided to assist the programmer in writing source-file entries.

The Instruction Set is presented in Chapter 5. The instruction set is organized according to six categories:

- Trace Data Format Specifiers
- Channel Format Specifiers
- Clock Specifiers
- Label Specifiers
- Status Specifiers
- Decode Arrays

In addition, Table 5-2 near the beginning of the chapter, lists the instructions by name and mnemonic, in an index for easy access. Each instruction begins on a separate page in a functional order within each category.

Chapter 5 also contains a description of the format of instruction statements, parameters, and predefined literals.

INSTALLING THE UDD-205

The UDD-205 is a software system which, when operated in conjunction with the K205 Logic Analyzer, displays the instructions in decoded form of predetermined portions of hardware or software operating in a target system.

It is the user's responsibility to devise methods of connecting the UDD-205 to the target system. Interface can be accomplished in different ways, depending on the needs of the user. The Logic analyzer can be connected to the target system directly using standard laboratory hardware, or using an interface board. Refer to the K205 User's Manual for instructions for connecting the Logic Analyzer to the target system.

Gould Inc. has a prototyping system available for connecting the UDD-205/K205 to a target system: the User Definable Interface (UDI). Contact a Gould Inc. representative for information regarding this product.

When configured and programmed, interface is accomplished. Once programmed, the database can be saved and reloaded for use when needed.

The UDD utility program (UUTIL) contains an editor and other functions that facilitate programming and using the UDD-205. The utilities are described in Chapter 6.

USING THE UDD-205

The user should be familiar with the utilities described in Chapters 6 and 7. Operation is treated in detail in Chapter 8, UDD Disassembler Operation, and briefly in Chapter 2, Getting Started.

UTILITIES

The UDD-205 utilities include:

- Editor - To enter and edit the database
- DOWNLINK - To down load a source file from an external computer into the UDD-205
- KLIST - To examine editor or list files on the K205 CRT
- TLIST - To examine editor or list files on the display screen of a computer attached to the Auxiliary port of the K205, or to print files
- SETTERM - To configure the editor to a specific external terminal
- UDDGEN - To generate a database for the UDD from the source file

GETTING STARTED

GENERAL

The functions that the user performs, using the UDD-205, parallels the components of the disassembler. These functions include:

- Creating the database source file
- Generating the database
- Operating the UDD-205

The functions are summarized in Figure 2-1

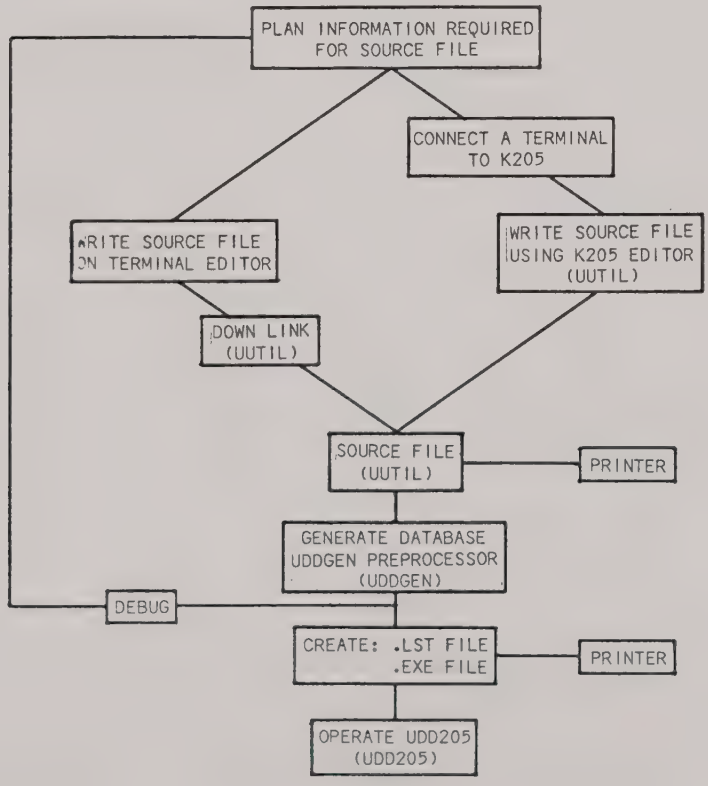


Figure 2-1. Functions of the UDD-205 System

Ideally, the user should be familiar with the entire content of this manual. Certain chapters are essential for different applications. Each section of this chapter includes a list of references that the user should have available to make best use of the UDD-205. In all instances, the user should also have available the manufacturer's documentation for the target system.

CREATE DATABASE SOURCE FILE

References

- UDD-205 USER'S MANUAL, Chapters 4, 5, 6, 7, and Appendices
- Target system manufacturer's documentation

Preparation

Preparation for creating a database using the UDD-205 Editor includes the following steps:

- Understand UDD theory, programming, and operating procedures.
- Determine the format of the disassembled instruction display line.
- Determine the display-screen formats of disassembled instructions. Four formats can be programmed to disable the display of selected portions of recorded data.
- Devise a strategy for developing the source file.
- Using Worksheets (Appendix D), create entries for the source file.

Prepare Source-File Entries

Prepare entries in the categories listed in Table 2-1. If more detailed information is required to write an entry, refer to the Instruction Set contained in Chapter 5. The items without an asterisk in Table 2-1 can be devised before constructing the opcode descriptors. Items with an asterisk may be devised in an interactive fashion as the opcode descriptors are developed. A brief example follows Table 2-1.

Table 2-1. Source-File Entries

ENTRY	FUNCTION
TGP	Trace groups identifies the input sections of the K205.
CLK	Clocks specifies portions of the Logic Analyzer clock screen. Refer to CLK in the Instruction Set, Chapter 5.
NAM	Name is the Data Display screen heading.
HDG	Heading is the Data Screen Disassembly header.
HLP	Help is the text of a message the user can recall by pressing the HELP key when no cursor is active in the Data Display screen.
MSK	Mask is the valid bit mask for the predefined status group TGP
CHK	Status check initializes the trace-alignment array from the status portion of the recorded data.
MMD	Miscellaneous display mode determines how the MISC group is displayed.
VTC	Valid trace codes determine the beginning of each display line after the first decode pass.
DMD	Display mode determines the format for displaying the machine information.
TCS	Trace control word status defines the name and values of the status selections available as quick keys on the Trace-Control screen.
OPC*	Opcode performs the instruction decoding.
OCL*	Opcode reference-number lists is a numbered word array of references to OPC specifiers.
INS*	Instruction blocks build disassembly display lines.
MCH	Machine blocks describe the data surrounding the processor instructions.
TPL*	Templates are numbered ASCII arrays containing the static portions of a display line for instructions or groups of instructions.
VLD*	Valid options lists are numbered word arrays of valid bit combinations for the field designated by the OPC specifier referencing the array.
GDP*	Group descriptors describe a changeable field in the disassembly display line.
GDL*	Group display lists are numbered arrays containing data used by several of the display types from the GDP specifier.

Example. This example shows how the first three opcode descriptors (OPC) were developed for the 8085 Example. Refer to the first two OPC WORKSHEETS in Appendix C, and lines 64, 65, and 66 in the Source File listed in Appendix B. This example will also lead to lines 96, 97, 104, 131, 139, 154, 166, 167, and 192 of the source file.

For the first OPC, 0, the programmer decided to describe a major group decode involving the two MSB bits. Index (IDX) 0 is selected because the instruction OPCODE is one byte in length. The mask (MSK) C0H limits decode to the two MSB bits of the instruction. All options are valid so VO_NUM is blank. There is no instruction block (INS), therefore there is no machine block (MCH) or instruction template (TPL), and IB_NUM is not used. The default OPC reference number (D_NUM) does not apply and is not used. Since the two MSB bits of the instruction designate four major groups, a number, 0, is assigned to an opcode descriptor list (OCL), and the entries 1, 2, 3, and 4 are entered in the list.

For the second OPC, 1, a similar analysis is used except the mask (07H) limits decode to the three LSB bits of the instruction. The OCL must differ and can contain eight elements. Therefore a new OCL_NUM is assigned, 1. In the example, the programmer eventually assigned opcode numbers 17 through 24 to OCL 1.

For the third OPC, 2, the mask allows the decode of the six LSB bits of the instruction. The programmer chose to let this represent HLT from the 8085 instruction set which is a unique instruction having a masked value of 110110 or 36H (the actual instruction code is 76H). This is a valid option so it is entered in a VLD specifier which is given a reference number, 0, and VO_NUM becomes 0. Since decode is now complete for the HLT instruction, an instruction block reference (INS) must be assigned. A template, 0, is assigned with a buffer index of 19 containing the mnemonic HLT for display. Since the HLT instruction is a one-byte instruction, machine block (MCH) 1 is assigned.

If the field being decoded does not match the element in the valid options array, decode branches to the default OPC reference number (D_NUM), 5. There is no OCL_NUM.

Building opcode descriptors continues until all members of the microprocessor instruction set have an opcode descriptor. A similar process is used to produce the group descriptors. When all the opcode descriptors have been built, the entries in the OPC and GDP WORKSHEETS are ready for entry into the source file. The programmer in the example grouped them by category for entry. Although they can be entered in any order, if the same entry is inadvertently entered twice, the result is an error condition.

Install Terminal for Data Entry

Two terminal configurations are possible:

- Connect a terminal to the K205 to make entries directly using the UUTIL editor.
- Make entries in an ASCII-based computer file, then connect the computer to the K205 to download file into the UDD-205 using the UUTIL DOWNLINK facility.

Make Source-File Entries

Enter data in any convenient order. Use the example 8085 database listed in Appendix B as a guide to the order and format of the entries. An explanation point introduces comments in the program. The UUTIL generates line numbers automatically. The two modes of data entry require slightly different procedures.

Direct Entry. When the terminal is connected to the K205, use the UUTIL editor to make entries directly into the source file. With the K205 powered up and the UDD-205 diskette in the disk drive, perform the following sequence of steps:

1. Connect terminal to the Auxiliary RS-232 port.

2. Select I/O screen 2, and set baud rate.

3. Select I/O screen 1 to boot the DOS.

4. Select recall function, Quick Key 1.

5. Select the UUTIL file and press F4 key to load the utility.

6. Select filename field, enter name of the new source file, and press F4 key to open file.

7. Control is transferred to the terminal. Type the source file.

DOWNLINK. When a source file is created with a remote computer editor, use the DOWNLINK facility to load entries into the K205. With the K205 powered up, the UDD-205 diskette in the disk drive, and the source file present on the remote computer, perform the following sequence of steps:

1. Connect remote computer to Auxiliary RS-232 port.

2. Select I/O screen 2, and set baud rate.

3. Select I/O screen 1 to boot the DOS.

4. Select recall function.

5. Select the UUTIL file and press F4 key to load the utility.

6. Press Quick Key 6 to select DOWNLINK.

7. Select filename field, enter name of the new source file, and press F4 key to execute down load.

Following either entry procedure, the source file is now stored in the UDD. A printer connected to the K205 Auxiliary port can be used to print a listing of the source file using the TLIST utility.

GENERATE DATABASE

References

- UDD-205 User's Manual: Chapter 7, Appendices I and J

Operations

Transform the source file into a database using the UDDGEN preprocessor utility by performing the following sequence of steps:

1. Select and load UDDGEN.
2. Select the source file.
3. Press the F4 key to initiate the process.

If the programming was successful, the preprocessor produces a .LST file and an .EXE file having the same name as the source file. If errors occur in the source file, the preprocessor produces a .LST file, but no .EXE file. It is necessary to debug the source file before the .EXE file can be created.

A printer connected to the K205 Auxiliary RS-232 port can be used to print the .LST or source files using the TLIST utility.

OPERATE THE UDD-205

References

- UDD-205 User's Manual: Chapter 8, Appendix E, F, G, H, J
- Target system manufacturer's documentation

Operations

With the K205 connected to the target system and powered up, make a recording of target system operation, then perform the following steps to disassemble the target system:

1. Place the K205 AC POWER switch to the ON position.
2. Gently insert the UDD-205 diskette into Drive A with the disk slot toward the rear of the unit and the label up. Lock the disk in place using the drive latch handle.
3. Similarly, insert the data diskette into drive B.
4. Press I/O and numeric 1 keys to display the Disk Screen Mode (DOS).
5. Press the F3 key to display directory B.
6. Press Quick Key 1 to select recall function.
7. Using cursor keys, select the UDD205.EXE file for loading. Press function key F4 to execute loading.

8. A HELP message appears. Press any key to eliminate the message; a file directory appears.

9. Using cursor keys, select the desired datafile for loading. Press function key F4 to execute loading.

10. Press the FORMAT and numeric key 4 to select the disassembler data format screen.

11. Press the DATA key to start disassembling.

THEORY OF OPERATION

GENERAL

This chapter describes the theory of operation of the User Definable Disassembler. To develop an efficient disassembler, the programmer should understand how the UDD manipulates recorded data. In addition to its use as a microprocessor disassembler, the UDD is also useful for decoding state machines, data buses, and communication interfaces.

Examples used to illustrate different functions are based on the 8085 eight-bit microprocessor. For that reason, recorded data samples are referred to as bytes. It is assumed that the logic analyzer record memory contains proper address, data, and status information from an 8085 based target system. Development of clocking and data-format setups is discussed in chapter 4.

A complete example of the 8085-microprocessor data base for the UDD-205 is shown in Appendix B. Worksheets used to build the data base are shown in Appendix C.

MICROPROCESSOR INSTRUCTION ARCHITECTURE

Knowledge of the structure of a microprocessor's instruction set is essential for the development of a database for the UDD. The more compact or regular the instruction set of the microprocessor, the smaller the disassembler database that is required. This section describes the architecture of a microprocessor from the viewpoint of a person designing a UDD database.

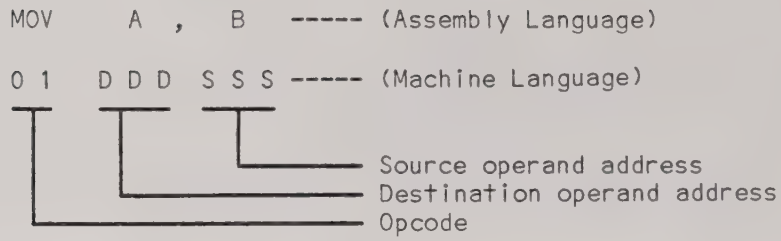
Instruction Opcode Description

A microprocessor instruction usually consists of an instruction operation code (opcode) followed by one or more operands. Some microprocessors extend the opcode into the operands. The opcode tells the microprocessor what operation to perform and what operands to use when performing that operation. Small fields within the opcode can be described as mini-opcodes that determine the instruction and the operands or operand addresses.

Example of a Simple Opcode

The 8085 MOV instruction, 0 1 D D D S S S, is an example of a combination of mini-opcodes that form a single-instruction opcode.

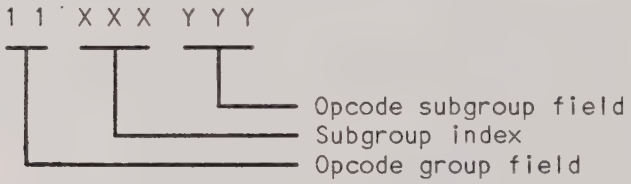
The eight-bit opcode is broken into three mini-opcodes, each specifying a distinct portion of the instruction. In addition, if the assembly language representation of the MOV instruction is examined, the mini-opcode bit fields can be seen as separate elements of the instruction. Refer to Example 3-1. In this example, the two-bit opcode field (01) defines a group of 64 instructions (eight source combinations times eight destination combinations).



Example 3-1. Simple Operation Code

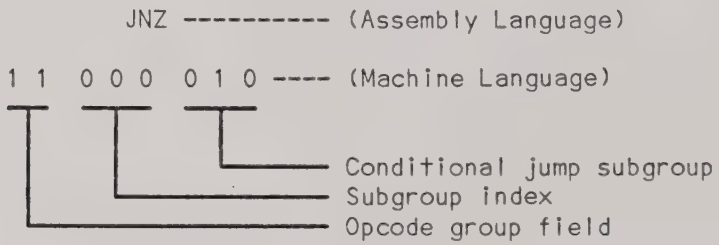
Example of a Complex Opcode

An example of a complex opcode is the 8085 Conditional Jump structure. Refer to Example 3-2. In this example, the two-bit field (11) specifies one of four possible groups of 64 instructions (eight subgroups of eight instructions each).



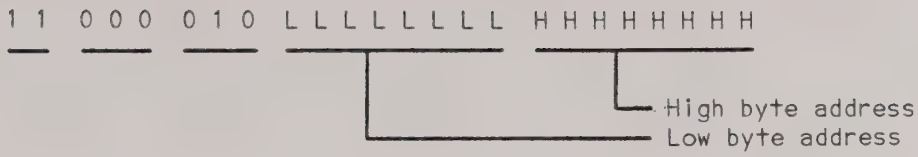
Example 3-2. Complex Operation Code

The 8085 conditional jump subgroup is contained in the XXX subgroup field. A specific instruction is then specified in the YYY subgroup index. Refer to the example in Example 3-3.



Example 3-3. Jump-If-Not-Zero Instruction Operation Code

A further complexity to the conditional jump instruction is the addition of an operand to the instruction, containing the destination address used if the condition is found true. Since the 8085 is an eight-bit microprocessor with a sixteen-bit address range, the instruction requires two additional bytes to contain the entire operand. Refer to Example 3-4.



Example 3-4. JNZ Operation Code with Two-Byte Address Operand

Most microprocessor instruction sets can be divided in this manner, providing a means of more efficiently utilizing the UDD-205. The next several chapters discuss the use of this decoding process to develop a custom database for the UDD-205.

OVERVIEW OF THE UDD

The UDD contains three modules: the Alignment, Decode, and Display modules. The Logic Analyzer takes data from the target system and sends them to the alignment module where the validity of each sample is determined, via the Decode module, and assigned a Trace Alignment Code. The data then enter the Display module where the samples are disassembled, via the Decode module again, and formatted for display. The modified samples are then returned to the logic analyzer and displayed. The process is described in detail in the following sections. An overview of the UDD structure is illustrated in Figure 3-1, and a description of data flow is illustrated in Figure 3-2.

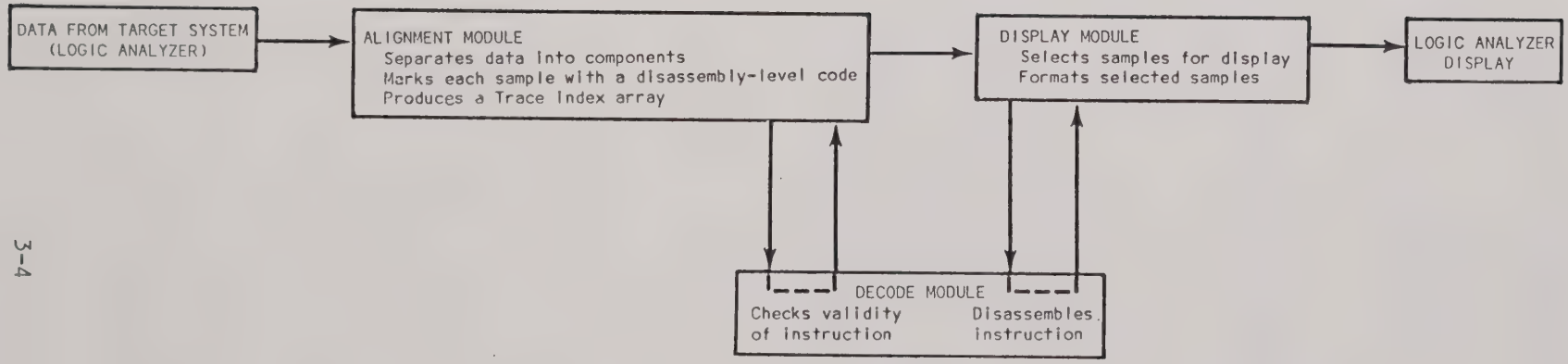


Figure 3-1. UDD Structure

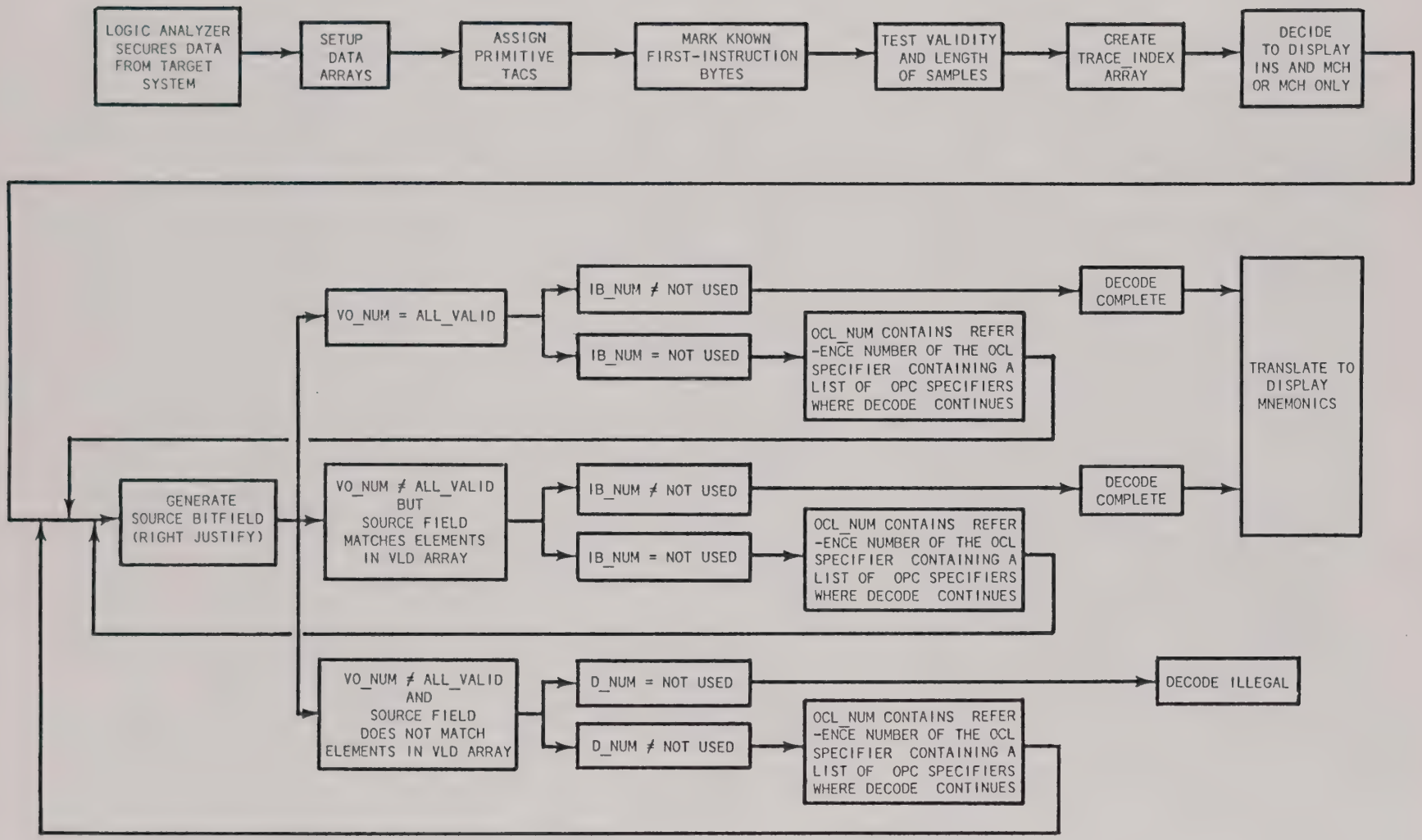

LOGIC ANALYZER SECURES DATA FROM TARGET SYSTEM
SETUP DATA ARRAYS
ASSIGN PRIMITIVE TACS
MARK KNOWN FIRST-INSTRUCTION BYTES
TEST VALIDITY AND LENGTH OF SAMPLES
CREATE TRACE_INDEX ARRAY
DECIDE TO DISPLAY INS AND MCH OR MCH ONLY
GENERATE SOURCE BITFIELD (RIGHT JUSTIFY)
VO_NUM = ALL_VALID
IB_NUM ≠ NOT USED
IB_NUM = NOT USED
OCL_NUM CONTAINS REFER-ENCE NUMBER OF THE OCL SPECIFIER CONTAINING A LIST OF OPC SPECIFIERS WHERE DECODE CONTINUES
DECODE COMPLETE
VO_NUM ≠ ALL_VALID BUT SOURCE FIELD MATCHES ELEMENTS IN VLD ARRAY
IB_NUM ≠ NOT USED
IB_NUM = NOT USED
OCL_NUM CONTAINS REFER-ENCE NUMBER OF THE OCL SPECIFIER CONTAINING A LIST OF OPC SPECIFIERS WHERE DECODE CONTINUES
DECODE COMPLETE
VO_NUM ≠ ALL_VALID AND SOURCE FIELD DOES NOT MATCH ELEMENTS IN VLD ARRAY
D_NUM = NOT USED
D_NUM ≠ NOT USED
OCL_NUM CONTAINS REFER-ENCE NUMBER OF THE OCL SPECIFIER CONTAINING A LIST OF OPC SPECIFIERS WHERE DECODE CONTINUES
DECODE ILLEGAL
TRANSLATE TO DISPLAY MNEMONICS

Figure 3-2. UDD Sample Processing

DATA ALIGNMENT

The logic analyzer must receive data from the target system before the disassembler performs its operations. The UDD first sends record data to the data alignment module. The purpose of the alignment module is to separate the record data into their component parts: address, data, and status; and then to mark each data sample with a code identifying the sample's data type.

Create Trace Arrays

The data alignment module creates three trace arrays containing the various segments of recorded data. A fourth array, Trace_align, is initially loaded with a direct derivation of the microprocessor status. A processor-read (PRD) trace status generates a PRD code in the Trace_align array; a processor-write (PWR) status generates a PWR code. These codes and others derived directly from the processor status are termed "primitive" Trace Alignment Codes (TACs). Some microprocessors provide status information that properly classify data samples without further decoding. Arrays representing the data components are illustrated in Example 3-5.

SAMPLE	TRACE_ ADDR	DATA	STAT	ALIGN	
1	0100	C2	42	PRD	(JNZ opcode)
2	0101	54	42	PRD	(JNZ operand)
3	0102	10	42	PRD	(JNZ operand)
4	1054	7E	42	PRD	(MOV A,M opcode)
5	3FFF	FF	42	PRD	(read data)
6	1055	36	42	PRD	(MVI M opcode)
7	1056	55	42	PRD	(MVI data operand)
8	3FFF	55	41	PWR	(write data)
9	1057	00	42	PRD	(NOP opcode)

Example 3-5. Sample Trace Arrays with Primitive TACs

Mark Known First-Instruction Bytes

The alignment module then steps through the trace arrays and marks all known first instruction bytes. The first processor-read found in the Trace_data array is assumed to be a first instruction byte. The address sequence of all processor-reads is tested for increments of more than one. If any adjacent addresses have a difference of more than one, the higher sample number of the two is assumed to be a first instruction byte. Each occurrence is marked in the Trace_align array with a "calculated" TAC of PF1 replacing the primitive TAC as illustrated in Example 3-6.

SAMPLE	TRACE_ ADDR	DATA	STAT	ALIGN	
1	0100	C2	42	PF1	(JNZ opcode)
2	0101	54	42	PRD	(JNZ operand)
3	0102	10	42	PRD	(JNZ operand)
4	1054	7E	42	PF1	(MOV A,M opcode)
5	3FFF	FF	42	PF1	(read data)
6	1055	36	42	PF1	(MVI M opcode)
7	1056	55	42	PRD	(MVI data operand)
8	3FFF	55	41	PWR	(write data)
9	1057	00	42	PRD	(NOP opcode)

Example 3-6. Sample Trace Arrays with Estimated First-Instruction Bytes

Note that the processor data reading in sample number 5 was interpreted as the first byte of an instruction. It is possible to get a false characterization of a sample at this stage. It may be mislabeled, but subsequent testing should correct the label.

Test the Validity and Size of Samples

Finally, the data alignment module sends all processor-reads, from one PF1 up to the next PF1, to the disassembler Decode module. The decode module returns information about the instruction to the alignment module: its validity, and if valid, its length and number of data-reads the instruction performs, if any. The alignment module then marks each byte with the appropriate calculated TAC: PF1 if the byte was an instruction first byte, PFX if the byte was an instruction extension byte, and DRD if the byte was a processor data-read caused by execution of the instruction.

The process is repeated using the next set of samples, starting with the next PF1, until the entire array has been decoded. The completed Trace_align array is illustrated in Example 3-7.

SAMPLE	TRACE_ ADDR	DATA	STAT	ALIGN	
1	0100	C2	42	PF1	(JNZ opcode)
2	0101	54	42	PFX	(JNZ operand)
3	0102	10	42	PFX	(JNZ operand)
4	1054	7E	42	PF1	(MOV A,M opcode)
5	3FFF	FF	42	DRD	(read data)
6	1055	36	42	PF1	(MVI M opcode)
7	1056	55	42	PFX	(MVI data operand)
8	3FFF	55	41	PWR	(write data)
9	1057	00	42	PF1	(NOP opcode)

Example 3-7. Sample Trace Arrays with Trace Align Array Completed

Create Trace_Index Array

The data alignment module produces a fifth array, the Trace_index array. Each element of the Trace_index array contains an index into Trace_align of each sample that begins a display line. Trace alignment codes that begin a display line are PF1, PRD, PWR, and DRD, PILL, and NODATA. Trace alignment code PFX does not begin a display line since multiple byte instructions are displayed on one CRT line. The completed Trace_align and Trace_index arrays contain alignment information that allows the disassembler Display module to operate. The completed Alignment-module output arrays are illustrated in Example 3-8.

SAMPLE	TRACE_ ADDR	DATA	STAT	ALIGN		INDEX
1	0100	C2	42	PF1	(JNZ opcode)	1
2	0101	54	42	PFX	(JNZ operand)	4
3	0102	10	42	PFX	(JNZ operand)	5
4	1054	7E	42	PF1	(MOV A,M opcode)	6
5	3FFF	FF	42	DRD	(read data)	8
6	1055	36	42	PF1	(MVI M opcode)	9
7	1056	55	42	PFX	(MVI data operand)	
8	3FFF	55	41	PWR	(write data)	
9	1057	00	42	PF1	(NOP opcode)	

Example 3-8. Sample Trace Arrays with Trace Align Array Completed

Primitive Identification of Other Status States

For microprocessors that provide more specific status information, the user can refine the information in the Trace_align array, during the early stage of disassembly. Assigning non-predefined TAC values to a microprocessor-cycle status causes the UDD to ignore those cycles when decoding instructions. Otherwise the Decode module must infer all PF1s from the structure patterns of the trace samples.

However, PF1 and PFX must both be identifiable if either is predefined for the first phase of decoding. That is, if PF1 is identifiable in the primitive state, PFX must also be identifiable, and conversely. If PF1 and PFX are not identifiable, they must both be assigned PRD.

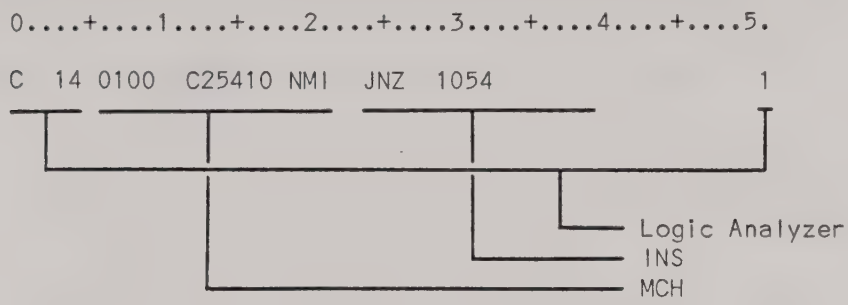
DISASSEMBLY DISPLAY

The purpose of the Display module is to determine which samples to display and how to display them. The Display module selects samples for display based on entries, in the Trace_index and Trace_align arrays, previously calculated in the Alignment module. The selected samples are placed in a disassembly display line formed by the Display module.

Characteristics of a Disassembly Display Line

There are three major sections to each disassembly display line. The first section, labeled FRAME, is generated by the logic analyzer and cannot be altered by the user. It consists of the sample and level display.

The other two sections are generated by the user database: the MACHINE BLOCK (MCH) and the INSTRUCTION BLOCK (INS). MCH contains the hardware information about a particular microprocessor instruction or data cycle: the address, object code, and status. INS contains the decoded mnemonic representing the object code. A disassembly display line is illustrated in Example 3-9.



Example 3-9. Disassembly-Line Structure

Select Samples for Display

The Display module uses the Trace_index number to identify which Trace_align entries to select. From the identified Trace alignment codes, the display module decides whether to include an INS and MCH, or just a MCH in the disassembly line. PF1 is the only trace alignment code that causes an INS display.

The Display module examines the data samples in the sequence determined in the Trace_index array. When a PF1 is encountered, the Display module sends the Trace_data sample associated with that TAC and up to the next nine Trace_data samples to the Decode module.

The Decode module, employing the same routine previously used by the Alignment module, generates a disassembled instruction. The process repeats until all data are processed.

INSTRUCTION DECODE

The decode process uses a modified tree search algorithm. The tree structure is embodied in the Operation Code (OPC) specifiers.

Characteristics of an OPC Specifier

Each OPC specifier provides four major pieces of information about the source data:

1. IDX and MASK determine the source data

   - IDX indexes into the Trace_data sent to the Decode module to determine which instruction sample is to be decoded.

   - MASK determines which portion of the Trace_data sample is to be the Source_data field.

2. VO_NUM determines if the source datum is valid

   - If VO_NUM = ALL VALID, every combination of the source field is considered valid.
   - If VO_NUM ≠ ALL VALID, the number in VO_NUM is a reference to to the Valid_options (VLD) specifier containing a list of valid combinations for the source field.

3. OCL_NUM and IB_NUM determine where to go if the source datum is valid

4. D_NUM determines where to go if the source datum is invalid.

Source Field Generation

The source data field is used for comparing decoded instructions with elements in the VLD array and for use with OCL_NUM. A process of right justification generates a source data field.

The "1" bits in the MASK are right justified resulting in a source field mask. The same transformations that produce the source field mask are applied to the Trace_data to produce the source fields.

Example 3-10 shows a Trace_data sample of 40H masked by MASK = 0C0H. When right justified, the MASK becomes a source field mask of 3H. Trace_data is transformed to a source field of 1H.

```
11000000  MASK becomes
00000011  source field mask.

therefore

01000000  Trace_data becomes
00000001  source field.
```

Example 3-10. Extract a Field Using Right Justification

Separated mask groups are combined into one group. Example 3-11 shows a Trace_data sample of 24H masked by MASK = 36H. When right justified and combined the MASK becomes a source field mask of 0FH. The Trace_data is transformed to a source field of 0AH.

```
00110110  MASK becomes
00001111  source field mask.

therefore

00100100  Trace_data becomes
00001010  source field.
```

Example 3-11. Extract a Field Using Right Justification, with Grouping

Determine if Decode Is Complete or Continues Branching

The Decode module determines whether decoding is complete or further decoding is necessary. The operation of IB_NUM and OCL_NUM is mutually exclusive. If IB_NUM is not used (IB_NUM = NOTUSED), OCL_NUM contains the reference number of a list of OPC reference numbers to which the branch continues. If IB_NUM is used (IB_NUM ≠ NOTUSED) OCL_NUM is ignored, and IB_NUM prevails. D_NUM and IB_NUM, or D_NUM and OCL_NUM, may coexist in the same OPC specifier.

Three possibilities obtain:

1. VO_NUM = ALLVALID leads to the following determinations:

   - IB_NUM ≠ NOTUSED, decode is complete. The reference number contained in IB_NUM is that of the INS (INSTR BLOCK) describing the decoded instruction mnemonic; or

   - IB_NUM = NOTUSED, OCL number contains a reference number of a list of OPC reference numbers to which branching continues.

2. VO_NUM ≠ ALLVALID, but the source field matches an element in the VLD array, leads to the following determinations:

   - IB_NUM ≠ NOTUSED, decode is complete. The reference number contained in IB_NUM is that of the INS (INSTR BLOCK) describing the decoded instruction mnemonic; or

   - IB_NUM = NOTUSED, OCL number contains a reference number of a list of OPC reference numbers to which branching continues.

3. VO_NUM ≠ ALLVALID, and the source field does not match an element in the VLD array, leads to the following determinations:

   - D_NUM = NOT USED, decode is illegal, the decode has failed and falls out or the tree and the user rediscovers the principle of gravity if sitting under the tree, or

   - D_NUM ≠ NOT USED, D_NUM contains reference number of OPC specifier into which branching continues.

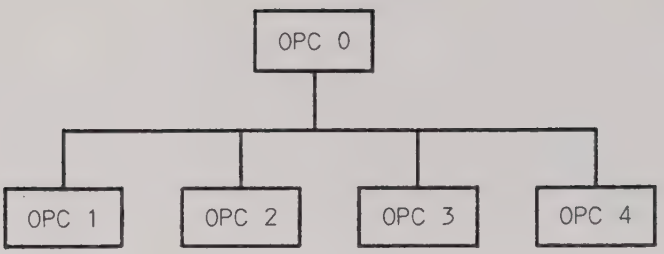
Branching

When conditions exist requiring further decoding, D_NUM will contain the reference number of the OPC specifier at which decode continues, and/or OCL_NUM will contain the reference number of a list of OPC specifiers. If the OCL_NUM is used, the source field is used as an index into the OCL_LIST to choose the correct OPC_Specifier for further decode branching. Examples 3-12 and 3-13, based on 8085 MOV Instruction 01 DDD SSS, represent branching selected through an OCL_NUM.

	IDX	MASK	VO_NUM	IB_NUM	D_NUM	OCL_NUM
OPC #0 --	0	0C0H	ALLVALID	NOTUSED	NOTUSED	1

OCL #1 -- 1, 2, 3, 4

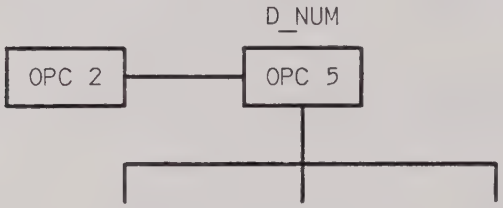
Example 3-12. UDD Source Code Representation of Tree Branching



Example 3-13. Graphic Representation of Tree Branching

Example 3-14 represents branching based on a D_NUM reference number. OPC #2 leads to OPC #5 which may branch again via OCL_NUM references.

	IDX	MASK	VO_NUM	IB_NUM	D_NUM	OCL_NUM
OPC #2 --	0	3FH	0	0	5	NOTUSED



Example 3-14. Default Tree Branching

Translating

When decode is complete, the reference number contained in IB_NUM is that of the Instruction Block (INS) describing the disassembled instruction .

Each INS contains a display Template (TPL) reference number, a MCH reference number, and a series of Group Descriptor (GDP) reference numbers.

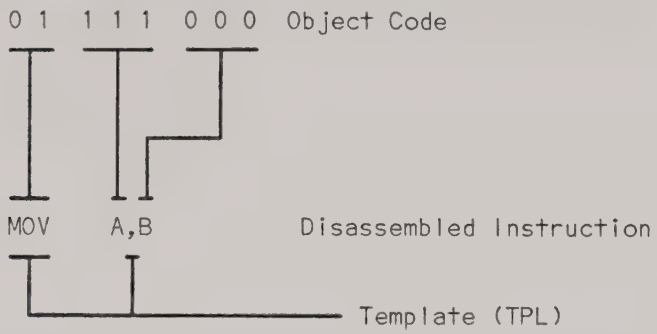
A GDP specifier describes a changeable field in the disassembly display line. Each specifier picks a sample word from the instruction sample array and isolates a field from that word in the same way that MASK and IDX determine source data in the OPC. The resulting field is displayed in a format determined by the rest of the elements of the GDP.

The GDP format elements describe where the formatted field will be displayed in the output buffer and in what form. OB_IDX specifies the position, and DSPLY_TYP specifies the form. Available forms are:

- Binary
- Octal
- Mnemonic
- Signed hexadecimal

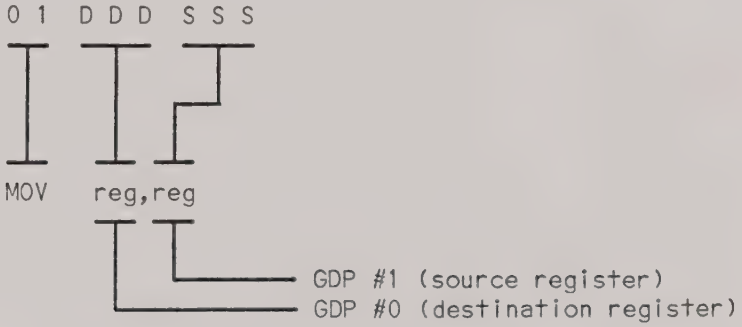
- Bit map
- Return sample count of instruction (not display type)
- Return read sample count of instruction (not display type)

The display TPL describes the nonchanging part of the disassembled instruction mnemonic. Refer to Example 3-15.



Example 3-15. 8085 MOV-Instruction Template.

Each GDP specifier describes a changeable portion of the display line. Refer to Example 3-16.



Example 3-16. 8085 MOV-Instruction GDP Assignment.

In addition, the MCH specifier referenced by the INS contains a list of GDP reference numbers describing the machine information part of the display line. Refer to Example 3-17.

Example 3-17. 8085 MOV-Instruction MCH GDP Assignment.

When the GDPs have been selected and placed in their proper locations, translation is complete. The translated instruction is returned to the display module which enters the translation in the display line. The Display module sends the disassembled instruction to the logic analyzer for display.

Build an OPC Decode Tree

The worksheets in appendix D can be used to build an OPC decode tree. They provide blanks for an OPC specifier and it's associated OCL (OPC_LIST), VLD list, INS, MCH, and instruction block TPL. The 8085 example in appendix C uses these worksheets to produce the sample disassembler database in appendix B.

Chapter 4

PROGRAMMING

GENERAL

The previous chapter on theory of operations is intended to give the user a theoretical understanding of how the UDD decodes microprocessor trace samples recorded on the logic analyzer. This chapter, describing the use of the UDD to build a disassembly database, is intended to bridge the gap between theory and operation of the UDD. The 8085 microprocessor is used in examples throughout this chapter.

The UDD is a very flexible system and it is impossible to prescribe a unique procedure for programming. Rather, elements required for building a disassembler database are presented along with examples to illustrate how a typical database was designed. The most important factor in programming the UDD is the knowledge and experience of the user in knowing what is required from the system.

This chapter relates to the 8085 examples contained in the appendices. Appendix B is a UDD database for the 8085 microprocessor. Appendix C contains completed worksheets used to build the GDP and OPC specifiers which, in turn, are used to develop the database. Appendix D contains blank worksheet masters from which to clone additional copies. These examples are referred to in this Chapter as the "8085 example."

There is some advantage in programming the UDD in the same order followed in this chapter. This order reflects the experience of the designers of the UDD-205.

TRACE GROUPS AND STATUS INFORMATION

Trace_Group Specifier

The TRACE_GROUP (TGP) specifier informs the UDD of the connections between the target microprocessor and the Logic analyzer. The microprocessor is divided into four signal groups:

1. The DATA group contains the actual instruction code or data resulting from an instruction.

2. The STATUS group contains signals which tell what type of data is on the DATA signal lines currently.

3. The ADDRESS group contains information about what memory location is associated with the instruction or data. Finally,

4. The MISC group contains information that is to be displayed, but doesn't affect the disassembly.

Signal Grouping

Each group must use sequential channels in increments of eight.

For example, if the address bus of a particular microprocessor has 12 lines, the user would have to assign 16 inputs (two sections) to the ADDRESS group. Group assignments may overlap sections if the clocking requirements of the two groups is the same.

In the 8085 example, the microprocessor uses section A-low for the DATA group, B-low and A-high for the-16 bit ADDRESS group, and C-low for the STATUS group, overlapping with the MISC group. In the example, the MISC group will display the interrupt status.

Status Information

Identify the status information provided by the microprocessor and assign a Trace Alignment Code (TAC) to each possible status condition.

Status Signal Assignments. In the 8085 example, the S0 and S1 status lines are placed on analyzer inputs C0 and C1, and the Input/Output Memory (IO/MEM-) line on input C2. The corresponding three bits in the STATUS MASK (MSK) specifier are set to inform the UDD that three microprocessor lines are to be used for status. Refer to Example 4-1.

C2	C1	C0	Logic Analyzer Inputs
IO/M-	S1	S0	Signal Names

Example 4-1. Status Signal Assignments

Status Check Array. The STATUS CHECK (CHK) array is used to tell the UDD what TAC is associated with each combination of the field selected by MSK. The three-bit field of STAT_MASK is used as an index into the CHK array. Refer to Example 4-2.

IO/M-	S1	S0	CHK	8085 Status Description
0	0	0	03H	HALT
0	0	1	PWR	MEMORY WRITE
0	1	0	PRD	MEMORY READ
0	1	1	PRD	OPCODE FETCH
1	0	0	03H	HALT
1	0	1	03H	I/O WRITE
1	1	0	03H	I/O READ
1	1	1	03H	INTERRUPT ACKNOWLEDGE

Example 4-2. 8085 Status Descriptions

Predefined TACs. Processor-write (PWR) and processor-read (PRD) are predefined Trace Alignment Codes (TACs). (See second, third, and fourth entries in Example 4-2.)

The predefined Trace Alignment Codes are used by the UDD to decide what trace data to decode as instructions and what trace data to display as non-instruction data. Any status information that may be available on the target microprocessor can be used to reduce chances of interpretation error.

In the 8085 example, The IO/MEM- line is used to distinguish between read/write cycles that are I/O related and cycles referencing memory. Since the I/O references are not instructions, assigning non-predefined TAC values to I/O cycle status causes the UDD to ignore those cycles when decoding instructions. If correct status information is available, even the non-primitive TAC's PF1 and PFX may be assigned.

Illegal Status. The predefined TAC 'PILL' is used to designate an illegal status condition. Note that PILL does not refer to an illegal instruction, but rather an illegal status condition. In the case of the 8085, all eight status conditions are defined, therefore PILL is unnecessary.

Define Names and Values of Status Selections. The STATUS TCW (TCS) specifier defines the names and values of the status selections available on the Trace Control screen of the Logic Analyzer. This feature allows the Logic Analyzer user to select the desired status pattern quickly without having to memorize bit patterns. In the example, the status combinations determined by the CHK array were chosen. Refer to Example 4-3.

```
TCS( 0, XXXXX010, 'READ    ')
TCS( 1, XXXXX001, 'WRITE   ')
TCS( 2, XXXXX110, 'INPUT   ')
TCS( 3, XXXXX101, 'OUTPUT  ')
TCS( 4, XXXXXX00, 'HALT    ')
TCS( 5, XXXXX011, 'FETCH   ')
TCS( 6, XXXXX111, 'ACK     ')
```

Example 4-3. Status Combinations

MACHINE_BLOCK (MCH)

To be useful, the disassembler must show the processor address, object code, and status used to derive the disassembled instruction. A Machine Block (MCH) describes how this information is to be formatted for each type of disassembler display line.

Display-Lines

The display line is 52 characters wide. It consists of a "frame" and a user-definable portion from the display buffer, the user display line. The user display line is 44 characters wide, and its own locations range from 0 to 43. Example 4-4 illustrates a display line with its enclosed user display line.

```
....+....1....+....2....+....3....+....4....+....5..  Display-Line Locations

      0....+....1....+....2....+....3....+....4....   User Display-Line Locations from the display buffer

C  14 _____________________________________________1
```

Example 4-4. Display Line Locations

The enclosed user display line contains a MCH and perhaps an Instruction Block (INS) containing the decoded mnemonic representing the object code of an instruction.

There can be several different display-line formats related to the number of trace samples required to obtain an entire instruction. For example, The 8085 microprocessor requires three different mnemonic display-line formats, corresponding to one, two, and three-byte instructions.

Each MCH entry contains a reference number and a list of Group Descriptor (GDP) reference numbers, normally one for every Trace Group displayed on the line. Note that the MCH does not contain any information for decoding the instruction mnemonic.

GDP Specifiers

The GDP specifiers describe that which is printed on the display line. They are common to both the MCH specifiers and the INS specifiers yet to be described.

In the 8085 example, MCH-zero describes an analyzer no-data sample. MCH- one through four describe the machine information for one-byte instructions, two-byte instructions, three-byte instructions, and data read/write cycles, respectively. MCH-one is presented as an example of how GDP specifiers are used to build a machine block.

Example. MCH-one describes the display line of a one-byte 8085 instruction. Example 4-5 illustrates how the display line is divided up for displaying the machine information for the one-byte instruction, where:

- 'aaaa' is the microprocessor address displayed in hexadecimal,
- 'oo' is the object code for the disassembled instruction in hexadecimal, and
- 'iii' is the position at which the interrupt status message will be displayed if an interrupt occurs during the execution of the instruction.

```
          0....+....1....+....2....+....3....+....4....

C  14 aaaa oo     iii                                1
```

Example 4-5. 8085 One-Byte Instruction Display Line

Note that the position of the instruction mnemonic display has not yet been defined in the above example.

MCH-one contains three GDP specifiers. GDP specifier-two, illustrating a typical specifier, displays the address group (aaaa) of the trace sample. A GDP worksheet is used as an aid to develop the GDP specifier. Refer to the worksheet in Appendix C illustrating GDP-two. Entries in the worksheet are explained as follows:

- The GDP reference number is 2
- IDX specifies ADDRESS group (040H) as the source of the displayed data
- MASK defines the size of the data to be displayed as 16 bits (0FFFFH)
- VO_NUM is ALLVALID since all combinations of the address field are displayed
- DSPLY_TYP 2, selects the unsigned hexadecimal display of the data defined by IDX and MASK. (Since 16 bits were specified by MASK, the number displayed is four digits in length.)
- DT_PAR is "NULL" since it is not required by the chosen DSPLY_TYP
- GDL_NUM is "NULL" since it is not used by the chosen DSPLY_TYP
- OB_IDX is 0 to start the display of the ADDRESS group at the zero output buffer index (user display-line position = 0)

The two other GDP specifier reference numbers in MCH-one, describing the interrupt status and object code displays (GDP specifiers 27 and 3 in the 8085 example), are not described in detail.

No-Data Situation

The no-data situation, MCH specifier 0, occurs when the user sets up Trace Control with auto-flush and records less than 515 samples. The non-recorded samples are referred to as no-data samples, and contain all zeros.

Identify Starting Display Line Cycles

Valid Trace Codes (VTCs) are defined as those types of microprocessor cycles that begin a display line in the disassembler. The codes themselves are selected from the TAC's defined by the CHK specifier and from the predefined UDD TAC's. In most cases the default for the VTC specifier can be used. This causes a display line to begin when any of the predefined TACs is encountered, with the exception of PFX which is not a VTC (PFX indicates an instruction-extension sample).

Since the 8085 status provides identification for five other types that should begin a display line, a code for these cycles (03H) is added to the example VTC specifier. Refer to Example 4-6.

```
VTC( 4, PF1, PRD, DRD, PWR, 3, NODATA)
```

Example 4-6. 8085 VTC Specifier, Full Display

The UDD is capable of having four VTC specifiers. In the preceding example, the first "4" in the string is the number of the VTC specifier. All VTC specifier arrays must be the same length.

Each VTC specifier is individually selectable by the UDD operator using the UDD Shift/S command. By specifying VTC arrays containing only the types of display lines desired, the data-base designer can provide the operator with different display formats. Use redundant TACs for place holders; the UDD recognizes only the first. For example, the VTC array illustrated in Example 4-7, causes a display format excluding all processor-data-read and processor-data-write cycles.

```
VTC ( 1, PF1, PF1, PF1, PF1, 3, NO_DATA)
```

Example 4-7. 8085 VTC Specifier, No Read/Write Cycles

Determine Format of Display Line

The DMD (Display Mode) specifier is used in conjunction with the VTC specifier to determine which MCH is to be used to format the display line. Each element of the DMD contains a MCH reference number which is to be used to format the display line for the corresponding VTC element. Refer to Example 4-8.

```
VTC( 4, PF1, PRD, DRD, PWR, 3, NODATA)
DMD(    0,   4,   4,   4,  4,   0,)
```

Example 4-8. 8085 VTC and Corresponding DMD Specifiers

The DMD element that corresponds to VTC element PF1 is ignored. Whenever a PF1 is encountered, the associated samples are decoded into instruction mnemonics and the MCH reference number is supplied by the decoder through the INS specifier.

DECODING AND DISPLAYING THE MICROPROCESSOR INSTRUCTION SET

The UDD decode tree structure described in Chapter 2 may be used in several ways to obtain a decode of the target processor. A straight table look-up method can be used for simple microprocessors and state machines. More complex targets require that the tree branches terminate (produce a decoded instruction or group of instructions) without decoding to completely unique bit combinations. A complete decode for such a complex target could use a large amount of memory which would not fully utilize the capabilities of the UDD.

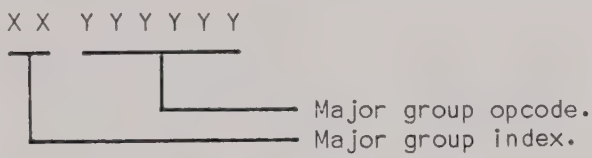
Three decoding methods follow that illustrate programming at different levels of complexity. Many other methods are possible. The methods suggested in this chapter are summarized in Table 4-1.

Table 4-1. Summary of Decode Methods

OPCODE STRUCTURE: XX YYY ZZZ			
METHOD	DEFINITION OF OPCODE BITS	FUNCTION	RESULT
1 (Direct decode)	XX - Major Group Index YYYZZZ - Major Group Opcode	Selects one of four major groups. Selects a look-up-table entry.	Direct selection of GDP entry
2 (Two-level decode)	XX - Major Group Index YYY - Destination Register ZZZ - Source Register	Selects one of four major groups. Used in a GDP Specifier to determine the destination register. Used in a GDP Specifier to determine the source register.	Selects two GDPs identifying source and destination registers for a given type of instruction.
3 (Three-level decode)	XX - Major Group Index ZZZ - Subgroup Index YYY - Opcode Subgroup Field	Selects one of four major groups. Selects up to eight instruction sub groups Used in a GDP Specifier to determine the mnemonic opcode.	Selects a GDP specifier which, in turn, decodes into an instruction mnemonic.

First Decode Method

Decoding the 8085 opcode yields four major opcode groupings delineated by the most significant two bits of the opcode, entries "XX" in Example 4-9.



Example 4-9. Opcode Structure

Each major group opcode, in turn, describes specific instruction groupings. If the major group opcode, YYYYYY, is completely decoded, it yields 64 separate instructions requiring 64 separate OPC specifiers. Example 4-10 illustrates 64 instructions and mnemonics for the 8085 MOV opcode (01YYYYYY).

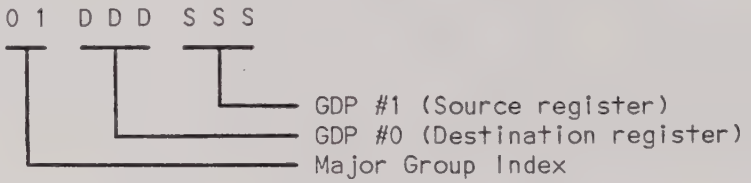
```
0 1  0 0 0  0 0 0    MOV   B , B
0 1  0 0 0  0 0 1    MOV   B , C
0 1  0 0 0  0 1 0    MOV   B , D
0 1  0 0 0  0 1 1    MOV   B , E
                .
                .
                .
0 1  1 1 1  1 0 0    MOV   A , H
0 1  1 1 1  1 0 1    MOV   A , L
0 1  1 1 1  1 1 0    MOV   A , M
0 1  1 1 1  1 1 1    MOV   A , A
```

Example 4-10. Decode Using Sixty-four Defined 8085 MOV Opcodes

Second Decode Method

The complete decode, however, reveals a pattern that allows the UDD to decode the MOV group more efficiently.

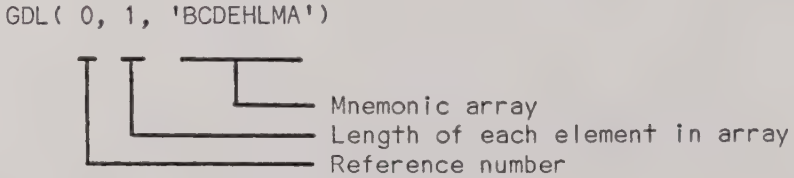
If the MOV group is declared to be completely decoded initially, the six least significant bits of the opcode can be broken into two three-bit fields, the most significant of which indicates the destination register and the least significant of which indicates the source register. Using this definition of the opcode, GDP specifiers can be called upon to print the register names using the source and destination fields as indices into a list of register names. This decoding method is illustrated in Example 4-11.



	IDX	MASK	VO_NUM	DT	DT_PAR	GDL_NUM	OB_IDX
GDP #0 --	0	038H	ALL_VALID	3	NOT_USED	0	24
GDP #1 --	0	07H	ALL_VALID	3	NOT_USED	0	26

Example 4-11. Decode Using Source and Destination Register Addresses

A Display Type (DT) of 3 causes the UDD to use the field generated by IDX and MASK as an index into the array referenced by GDL_NUM. In this case, GDL #0 (referenced by GDL_NUM = 0) contains a list of register names for the 8085. The GDL is illustrated in Example 4-12.

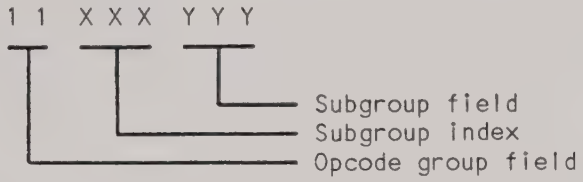


Example 4-12. GDL Specifier for Register Mnemonics.

For another example, if the fourth major group (11XXXYYY) is completely decoded, it yields 64 separate instructions. Again, the complete decode reveals a pattern that allows the UDD to more efficiently decode the group.

The six least significant bits break down into two three-bit fields. The least significant of fields (Subgroup Bit field) can be used in an OPC specifier to define eight subgroups of eight instructions each. Each of these subgroups can be declared completely decoded, with the final three bits (Subgroup Index) used in a GDP specifier to determine the mnemonic opcode.

Examples 4-13 through 4-16 illustrate the third decode method starting with the instruction format, and ending with a third-level decoded instruction. Example 4-14 shows the four opcode groups, referenced by OPC #0, and eight subgroups, referenced by OPC #4. Example 4-15 illustrates the decode tree structure.

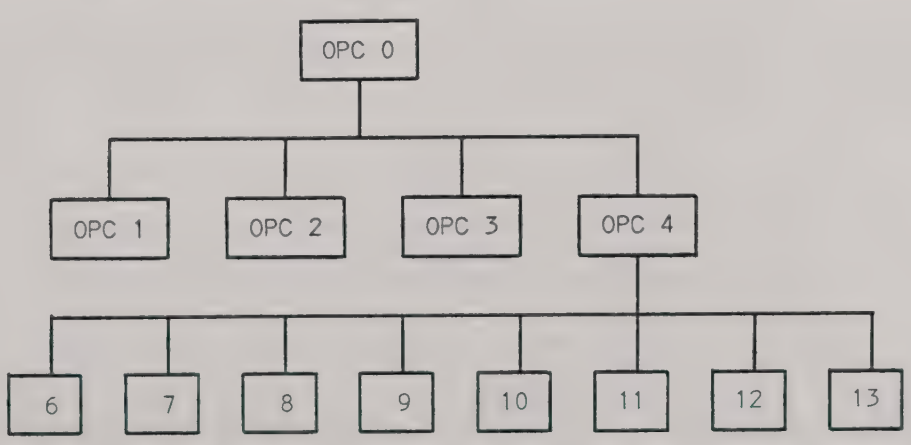


Example 4-13. 8085 Method-Three Instruction Format.

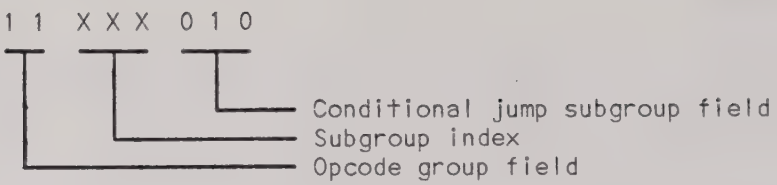
	IDX	MASK	VO_NUM	IB_NUM	D_NUM	OCL_NUM
OPC #0 --	0	0C0H	ALL_VALID	NOT_USED	NOT_USED	0
OPC #4 --	0	007H	ALL_VALID	NOT_USED	NOT_USED	2

OCL #0 -- 1, 2, 3, 4
OCL #1 -- 6, 7, 8, 9, 10, 11,, 12, 13

Example 4-14. Decode Using OPC and GDP Specifiers



Example 4-15. Method-Three Decode Tree.

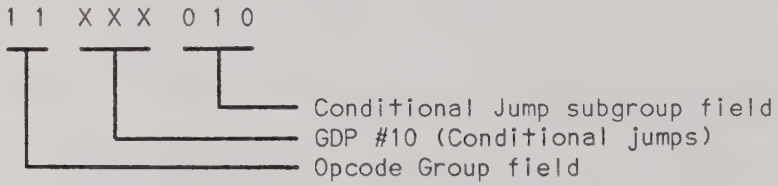


Example 4-16. Method-Three Conditional Jump Instruction Decoded

INSTRUCTION BLOCK (INS)

In the third decode method, once the opcode has been decoded down to the third level (OPC #8), the instruction can be considered completely decoded. This leaves bits three, four, and five (XXX) undecoded.

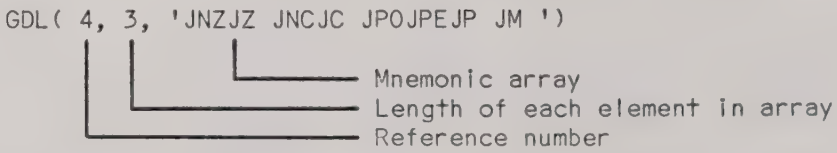
OPC #8 then references INS #4 (IB_NUM = 4). INS #4 contains the reference number of template #2, which is blank since there are no common display elements in the Conditional Jump Group, MCH #3 (three-byte instruction), and GDP specifiers 10, 11, and 12. GDP specifier #10 decodes the remaining three undecoded bits into the instruction mnemonic as follows. Refer to Example 4-17.



	IDX	MASK	VO_NUM	DT	DT_PAR	GDL_NUM	OB_IDX
GDP #10 --	0	038H	ALL_VALID	3	NOT_USED	4	19

Example 4-17. GDP from a Third Level Decode

A DT of 3 causes the UDD to use the field, generated by IDX and MASK, as an index into the array referenced by GDL_NUM. In this case, GDL #4 (referenced by GDL_NUM = 4) contains a list of Conditional Jump instruction mnemonics for the 8085. Refer to Example 4-18.



Example 4-18. GDL Specifier for Conditional Jumps

GDPs #11 and #12 display the two bytes of the Conditional Jump Operand.

INSTRUCTION SET

GENERAL

This chapter provides detailed information regarding the database instructions available to the UDD database designer. Each instruction is listed, starting on a separate page, and contains the following information:

- Statement Identifier
- Description
- Format
- Default

Statement Identifier:

The statement identifier appears in bold print at the upper left corner of the page starting each instruction. It is the key word recognized by the UDD. The identifiers must be entered exactly as shown. Characters can be entered using upper or lower case letters, or a mixture of both.

Description:

The description statement describes the UDD instruction statement, and contains conditions which govern the instruction's use.

Format:

The format states the syntax of the instruction statement, followed by a brief description of each of the instruction's parameters.

Default:

The default statement states the default used by the UDD if a statement is not entered by the user. Some UDD statements do not have corresponding defaults.

INPUT-FILE CONSTRUCTION

The preprocessor input file is generated using the editor supplied with UUTIL or an external editor compatible with the UUTIL editor. Externally generated files may be loaded into the K205 through the DOWNLINK utility found in UUTIL.

The UDD programmer (user) defines the format of the UDD display lines 5 to 25, and columns 8 to 51, inclusive. Each target processor instruction can be displayed on one or two CRT lines. The output buffer referred to in the following section is an analog of the user-defined portion of the UDD display line, indexed from 0 to 43. In actual use, the user has two buffers available designated OB_1 and OB_2, describing the first and second lines of an instruction, respectively.

The data base is entered in a freeform format with data structures similar to high-level language data declarations.

ASCII strings are surrounded by single quotes. Numeric constants are in decimal unless followed by a base specifier from the following list:

H = Hexadecimal

B = Binary

Q = Octal

T = Decimal (default)

Hexadecimal base constants must be preceded by a zero if the first character is in the range A to F.

Parameters are specified as follows:

Byte = eight bits

Word = sixteen bits

Double words = thirty-two bits

The preprocessor ignores all characters on a line following an exclamation-point (!) character, unless enclosed by single quotation marks.

The predefined literals and their associated values are listed in Table 5-1. The literals can be used as parameters to applicable specifier statements.

Table 5-1. Predefined Literals

LITERAL	VALUE
PILL	00H
PRD	01H
PWR	02H
PF1	40H
PFX	41H
DRD	42H
NODATA	80H
ALLVALID	0FFFFH
NOTUSED	0FFFFH
NULL	00H
+	1
-	0

LITERAL	VALUE
AL	0
AH	1
BL	2
BH	3
CL	4
CH	5
DATA	0
STATUS	1
ADDRESS	2
MISC	3

INSTRUCTION INDEX

The UDD instructions are listed in Table 5-2 which serves as an index to this chapter.

Table 5-2. Index to UDD Instructions

CHK	Status Check	5-23
CLK	Clocks	5-12
DMD	Display Mode	5-25
GDL	Group Display List	5-38
GDP	Group Descriptor	5-33
HDG	Heading	5-17
HLP	Help	5-19
INS	Instruction Block	5-31
MCH	Machine Block	5-32
MMD	Miscellaneous Display Mode	5-27
MSK	Status Mask	5-22
NAM	Name	5-16
OCL	Opcode List	5-36
OPC	Opcode	5-30
POL	Polarity	5-10
TCS	Trace Control Word, Status	5-26
TGP	Trace Group	5-6
THR	Thresholds	5-8
TPL	Template	5-37
VAR	Variable Threshold	5-9
VER	Version	5-18
VLD	Valid Options	5-35
VTC	Valid Trace Codes	5-24

TRACE DATA FORMAT SPECIFIERS

General

The UDD separates trace data into four groups: data, address, miscellaneous, and status. Each group must use sequential channels with increments of eight input channels. If applicable, the groups may overlap.

There is only one Trace Data Format Specifier:

- TGP

TGP

Description:

TGP (Trace Group) specifies the group sizes and input sections. The Logic Analyzer input sections are numbered according to the following list.

A lo - 0 (AL)
A hi - 1 (AH)
B lo - 2 (BL)
B hi - 3 (BH)
C lo - 4 (CL)
C hi - 5 (CH)

Format:

TGP (GROUP_NO, SECTION [,SECTION...])

where:

GROUP_NO = Represents the UDD group number.

0 - DATA
1 - STATUS
2 - ADDRESS
3 - MISC

SECTION = Represents the Logic Analyzer section number to be used.

A maximum of four input sections (thirty-two bits) may be specified for groups 0 and 2, while groups 1 and 3 may contain a maximum of one section. Groups 1 - 3 may be eliminated by specifying a GROUP_NO without a SECTION. If group 1 (status) is eliminated, every sample is assumed to be a processor read. If group 2 (address) is eliminated, every sample is assumed to be in sequential ascending order.

Default:

TGP (DATA, AL)
TGP (STATUS, BH)
TGP (ADDRESS, AH, BL)
TGP (MISC)

CHANNEL FORMAT SPECIFIERS

General

Each input channel in the Logic Analyzer may be assigned a polarity (normal or inverted) with which the data are to be interpreted. In addition, each Logic Analyzer input section may be recorded with TTL level threshold, ECL level threshold, or Variable level threshold. The clocks may be assigned a threshold as a group independent of the data inputs.

Channel format specifier instructions include the following:

- THR
- VAR
- POL

THR

Description:

THR (Thresholds) specifies the threshold level for each group including the clocks. The code for selecting a threshold is as follows.

0 - TTL
1 - ECL
2 - Variable threshold A
3 - Variable threshold B

Format:

THR (CLK, ALO, AHI, BLO, BHI, CLO, CHI)

where:

Each parameter contains the code for the threshold to be used in the corresponding section.

Default:

THR (0, 0, 0, 0, 0, 0, 0)

VAR

Description:

VAR (Variable) contains the code for the threshold level to be used if the variable threshold option is selected for any of the input sections.

Format:

VAR (REF, POL, THRESHOLD_LEVEL)

where:

REF = Indicates the variable threshold being programmed; A or B.

POL = Indicates polarity of the Threshold Level, '+' for positive, '-' for negative.

THRESHOLD_LEVEL = a constant between 0 and 999 where each step represents a hundredth of a volt.

Default:

VAR (A, +, 999)
VAR (B, -, 999)

POL

Description:

POL (Polarity) specifies the input channel polarity for each of the 48 input channels of the Logic Analyzer.

Format:

POL (A0, A1, A2, ... CD, CE, CF)

where:

For each input parameter, '+' indicates positive polarity , and '-' indicates negative (inverted) polarity.

Default:

POL (+, +, +, ... +, +, +) (All positive polarity).

CLOCK SPECIFIERS

The Logic Analyzer defines a separate clock equation for the master section and each of the three input sections, A to C. A separate enable clock equation is also defined for each section. Each equation may incorporate a combination of up to six clocks. The Boolean equation is as follows:

(CJ * BJ * AJ) + (CK + BK + AK) for clock equations
(CR * BR * AR) + (CS + BS + AS) for enable clock equations

Each clock can be normal (rising edge) or inverted (falling edge), or not used.

The demux capability causes the lower eight bits of data in each section to be duplicated on the upper eight bits.

There is only one clock specifier instruction:

- CLK

CLK

Description:

CLK (Clocks) specifies the user defineable portions of the Logic Analyzer clock screen for the custom disassembler.

Format:

```
CLK (
     CLK_SOURCE,
     CLK_INTERVAL,
     CLK_DECADE,
     S_A, S_B, S_C,
     S_M,
     M_CJ, M_BJ, M_AJ, M_CK, M_BK, M_AK,
     SA_CJ, SA_BJ, SA_AJ, SA_CK, SA_BK, SA_AK,
     SB_CJ, SB_BJ, SB_AJ, SB_CJ, SB_BK, SB_AK,
     SC_CJ, SC_BJ, SC_AJ, SC_CJ, SC_BK, SC_AK,
     EA_CR, EA_BR, EA_AR, EA_CS, EA_BS, EA_AS,
     EB_CR, EB_BR, EB_AR, EB_CS, EB_BS, EB_AS,
     EC_CR, EC_BR, EC_AR, EC_CS, EC_BS, EC_AS,
     DS_A, DS_B, DS_C,
     SL_AL, SL_AH, SL_BL, SL_BH, SL_CL, SL_CH,
     SS_A, SS_B, SS_C)
```

where:

CLK_MODE = K205 clock source. The following are legal values:

0 = Standard
1 = Advanced

CLK_INTERVAL = Clock interval. Used with CLK_DECADE to determine internal clock period. The following are legal values for INT:

1	-	0,	Steps of 1
10	-	90,	Steps of 10
100	-	900,	Steps of 100

CLK_DECADE = Clock decade. Used with CLK_INTERVAL to determine internal clock period. The following are legal values for DEC.

0 = milliseconds.
1 = microseconds.
2 = nanoseconds.

CLK_INTERVAL and CLK_DECADE together determine the internal clock period for the Logic Analyzer. For example, if CLK_INTERVAL = 50 and CLK_DECADE = 2, then the internal clock period is 50 nanoseconds. Minimum allowable internal clock period is 20 nanoseconds.

S_A, S_B, S_C, = Clock source. Each member selects the clock source for the A, B, and C sections, respectively. The select code is as follows:

0 = Internal
1 = External
2 = 10 nanoseconds
3 = Same as master

S_M = Clock source. Selects the clock source for the master clock. The select code is as follows:

0 = Internal
1 = External

M_CJ, M_BJ, M_AJ, M_CK, M_BK, M_AK = Master clock expression. The select code is as follows:

0 = Clock is not used
1 = Rising edge
2 = Falling edge

SA_CJ, SA_BJ, SA_AJ, SA_CK, SA_BK, SA_AK = Section A clock expression. Same as Master clock expression.

SB_CJ, SB_BJ, SB_AJ, SB_CJ, SB_BK, SB_AK = Section B clock expression. Same as Master clock expression.

SC_CJ, SC_BJ, SC_AJ, SC_CJ, SC_BK, SC_AK = Section C clock expression. Same as Master clock expression.

EA_CR, EA_BR, EA_AR, EA_CS, EA_BS, EA_AS = Section A enable clock expression. Same as Master clock expression.

EB_CR, EB_BR, EB_AR, EB_CS, EB_BS, EB_AS = Section B enable clock expression. Same as Master clock expression.

EC_CR, EC_BR, EC_AR, EC_CS, EC_BS, EC_AS = Section C enable clock expression. Same as Master clock expression.

DS_A, DS_B, DS_C = DEMUX_SEL. Each member selects the demux mode for the corresponding section. The select code is as follows:

0 = Deselect demux mode
1 = Select demux mode

SL_AL, SL_AH, SL_BL, SL_BH, SL_CL, SL_CH = Sample/latch. The select code is as follows:

0 = Sample only
1 = Latch data
2 = Glitch mode

SS_A, SS_B, SS_C = Store/sample. The select code is as follows:

0 = Sample
1 = Store

Default:

```
CLK (
     1,
     20,
     2,
     1, 1, 1
     1,
     0, 0, 1, 0, 0, 0,
     0, 0, 0, 0, 0, 1,
     0, 0, 0, 0, 0, 1,
     0, 0, 0, 0, 0, 1,
     0, 0, 0, 0, 1, 0,
     0, 0, 0, 0, 0, 0,
     0, 0, 0, 0, 0, 0,
     0, 0, 0,
     0, 0, 0, 0, 0, 0,
     0, 0, 0,)
```

LABEL SPECIFIERS

Description:

This section describes constants for Logic Analyzer disassembler mode labels, headers, and cursor movement in the Trace Control and Format screens in addition to the Data Display screen.

The label specifier instructions include the following:

- NAM
- HDG
- VER
- HLP

NAM

Description:

NAM (Name) is the Data Display screen heading.

Format:

NAM (string)

where:

'string' is an ASCII string of up to 15 characters enclosed in single quotes.

Default:

NAM ('UDD205')

HDG

Description:

HDG (Heading) is the Data Screen disassembly header.

Format:

HDG (string)

where:

'string' is an ASCII string of up to 44 characters enclosed in single quotes.

Default:

HDG ('ADDR OBJECT STATUS MNEMONIC ')

VER

Description:

VER (Version) is the UDD database version number.

Format:

VER (string)

where:

'string' is an ASCII string of up to 5 characters enclosed in single quotes.

Default:

VER ('1.xx');

where:

'xx' is replaced by the version number of the Preprocessor output file containing the data base.

HLP

Description:

HLP (Help) is the text of the UDD non-edit help message obtained when the UDD user presses the HELP key.

Format:

HLP (string_1, string_2, ... string_8)

where:

Each 'string' is an ASCII string of up to 52 characters enclosed in single quotes.

Default:

HLP (' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ')

STATUS SPECIFIERS.

General

This section deals with the recorded processor status and the alignment status assigned to each recorded sample as a result of the recorded status and internal processing.

Trace Alignment Codes (TACs) are used by the UDD in determining the status of trace data during the decode process. Certain codes are pre-defined by the UDD, while others are left open for use by the database designer. Codes below 40H are 'primitive' alignment codes which are assigned as a direct interpretation of the recorded status (see CHK). Codes 40H and higher are used by the UDD when calculating the beginning of an instruction sequence. These codes may also be assigned in the same manner as the primitive alignment codes if the target processor provides sufficient status information. The list of alignment codes used by the UDD is as follows.

00H - PILL, Illegal status.
01H - PRD, Processor read.
02H - PWR, Processor write.
03H - 3FH, User defined.
40H - PF1, Instruction first sample.
41H - PFX, Instruction extension sample.
42H - DRD, Known Data Read
80H - NODATA, Non-recorded data.

Codes 03H - 3FH may be defined by the database designer as needed. For example, the target processor may have status lines which indicate a DMA read, in which case an alignment code may be assigned to that status so that it will not be interpreted as a processor read. Codes are defined by associating a new code with a status value in the CHK array.

The status specifier instructions include the following:

- MSK
- CHK
- VTC
- DMD
- TCS
- MMD

MSK

Description:

MSK (Status Mask) contains the valid bit mask for the status group defined earlier. Only those bits that are used for trace alignment status definition should be masked. Other status bits that must be displayed but are not used in the determination of alignment status are handled in the MCH and GDP specifiers.

Format:

MSK (MASK)

where:

MASK = Byte. A one in a bit position indicates that the bit is a valid status bit, a zero indicates that the bit is not used.

Default:

MSK (0FFH), if Status Group defined (TGP(STATUS,xx))
MSK (00H), If Status Group not defined (TGP(STATUS))

CHK

Description:

CHK (Status Check) is used to initialize the trace alignment array from the status portion of the recorded data. Each entry corresponds to a bit pattern from the status byte as masked by MSK; if MSK has five bits set, there must be 32 entries in CHK. Each element contains a TAC that corresponds to the status bit pattern.

Format:

CHK (CODE_0, CODE_1 [, CODE_2, ... CODE_255])

where:

CODE_xx = Status code in range 00H - 042H.

Default:

Array must be specified by the user, if Status Group defined.

CHK (PRD) if Status Group not defined.

VTC

Description:

VTC (Valid Trace Code) is used after the first decode pass to determine the beginning of each line to be displayed during the second decode pass. If the trace alignment code assigned to a sample matches an entry in VTC, that sample becomes the beginning of a display line. This array is only slightly different than the CHK array in that it does not include status types such as PFX, and possibly special alignment codes that the data-base designer does not wish to be displayed.

There are four VTC arrays which correspond to soft keys F1 - F4. This allows the user to specify display modes that do not include certain types of processor cycles by entering a VTC specifier that does not contain the alignment code in question. All four VTC arrays must be the same length as VTC reference number one. If an alignment code is to be left out, it must be replaced by a duplicate of an already entered code.

Format:

VTC (REF, CODE_0 [, CODE_1, ... CODE_41])

where:

REF = The reference number of the VTC array, becomes the soft-key number used to initiate the SCREEN display mode command. Range is 1 - 4.

CODE_xx = Alignment code in ranges 00H - 03FH, 40H - 42H, and 80H.

Default:

VTC (1, PILL, PILL, PILL, PILL, PF1, NODATA)
VTC (2, PILL, PRD, PWR, DRD, PF1, NODATA)
VTC (3, PILL, PRD, PWR, DRD, PF1, NODATA)
VTC (4, PILL, PRD, PWR, DRD, PF1, NODATA)

DMD

Description:

DMD (Display Mode) is used to determine the format for displaying the machine information for the disassembled line. Elements in this array have a direct correspondence to elements of the VTC array. When displaying a disassembled line, the VTC index for the alignment code is used as an index into the DMD array. The resulting value is the reference number of the MCH specifier that determines the display format.

Format:

DMD (CODE_0 [, CODE_1, ... CODE_255])

where:

CODE_xx = MCH reference for the alignment code in the corresponding entry of VTC. For PF1, the DMD code is ignored by the UDD. The MCH reference for these codes is generated by the appropriate INS specifier.

Default:

DMD (0,0,0,0,0,0)

TCS

Description:

TCS (Trace Control word, Status) defines the names and values of the status selections available as quick-keys when altering the Trace Control word Status field.

Format:

TCS (REF, STATUS_CODE, string)

where:

REF = The number of the TCS array becomes the quick-key number used to reference the status definition in the Trace Control screen. Range is 0 - 15 (00H - 0FH).

STATUS_CODE. This is a special representation of the eight bits of the STATUS group. One character is needed for each bit. The valid characters are 0, 1, and X, equivalent to logic zero, logic one, and "don't care".

'string' is an ASCII string of up to 8 characters representing the name of the defined status pattern enclosed in single quotes.

Default:

```
TCS( 0, XXXXXXXX, 'XXXXXXXX')
TCS( 1, XXXXXXXX, 'XXXXXXXX')
               .
               .
               .
TCS( 15, XXXXXXXX, 'XXXXXXXX')
```

MMD

Description:

MMD (Miscellaneous Display Mode) determines how the MISC group is displayed. This instruction allows the user to 'AND' or 'OR' all MISC group samples in a multi-sample processor instruction together so that interrupts or other MISC Group status changes that occur in the middle of an instruction are displayed.

Format

MMD (MODE, XOR)

where:

MODE =

0 = no action
1 = 'AND' samples
2 = 'OR' samples

XOR = Byte. This byte is exclusive-or'd with the MISC group data before the 'AND' or 'OR' operation is carried out. This allows the user to coerce a status bit that is the wrong polarity for the selected operation.

Default

MMD (0, 0)

DECODE ARRAYS

This section describes the arrays used to disassemble the target processor. The user should view the processor instruction as an array of sample words. The sample word width was determined in section 4.1 as TGP zero (DATA).

The Decode Arrays instructions include the following:

- OPC
- INS
- MCH
- GDP
- VLD
- OCL
- TPL
- GDL

OPC

Description:

OPC (Opcode) performs the major instruction group decoding for the UDD. It is a modified tree structure. Each entry in OPC isolates a field in the instruction. A field may be from one to 16 bits in width and may be non-contiguous. That is, multiple fields defined by the bit mask will be joined by the UDD into one large field. Once a field has been defined, it may be used to branch further into the OPC tree or, if the instruction can be grouped (defined) at this point, the field may be used as an index in the next level of decode, the INS.

Format:

OPC (REF, IDX, MSK, VO_NUM, IB_NUM, D_NUM, OCL_NUM)

where:

REF = The number of the OPC specifier, used by the OPC D_NUM and OCL_NUM to branch further into the OPC tree.

IDX = Index into instruction sample array of word containing the opcode field to be masked. Range is 0 to 9.

MSK = Double word. Contains the mask for the opcode field in instruction. A one in any bit location signifies that the corresponding bit in the instruction sample is to be used as part of the opcode field. Mask data are right-justified and separate mask groups are combined (see text). Any combination of up to 16 bits may be selected.

VO_NUM = Reference number of the VLD specifier containing the valid opcodes for this field, or "ALL VALID" if all codes are valid.

IB_NUM = Reference number of the INS specifier for this opcode group, or "NOT USED" if group is not unique at this level.

D_NUM = The default OPC reference number used if the field value does not match an element of the VLD array. Only used if VO_NUM is NOT "ALL VALID". If VO_NUM is "ALL VALID", or the field combination that does not match is an illegal instruction, D_NUM should be set to "NOT USED".

OCL_NUM = Reference number of the OCL array containing the list of OPC reference numbers for each member of current opcode group, or "NOT USED" if group is unique at this level.

The index into the indicated OCL array is developed from the field. If VO_NUM is "ALL VALID", then the OCL index is equivalent to the value of the field. Otherwise, the OCL index is equivalent to the index of the VLD array element that matches the value of the field.

Default:

No default. This specifier must be entered by the user.

INS

Description:

INS (Instruction Block) is used to build a disassembly display line. It consists of a reference number, a TPL reference number, and a list of GDP (Group Definition) specifier reference numbers. The template contains the non-changing part of the display for a particular opcode group. The GDP specifiers define the parts of the display line that differ for the different members of an opcode group.

Format:

INS (REF, T_NUM, M_NUM, GD_NUM [, GD_NUM ..., GD_NUM])

where:

REF = The number of the INS specifier, used by the OPC specifier to reference the INS.

T_NUM = Reference number of the TPL array containing the basic form of assembly display line (commas, brackets, etc).

M_NUM = Reference number of the MCH specifier containing the machine information format for this instruction block.

GD_NUM = Reference number of a GDP specifier used to display all or part of the disassembled instruction.

Default:

No default. This specifier must be entered by the user.

MCH

Description:

The MCH (Machine Block) describes the data surrounding the processor instruction, ie, the address, object code, and status. The data displayed are the data direct from the TGP definitions. The ADDRESS and STATUS groups display data from the first sample only of a multi-sample instruction. The MISC group may optionally display all samples of an instruction OR'd together or AND'd together (see MMD).

Format:

MCH (REF, GD_NUM [, GD_NUM ..., GD_NUM])

where:

REF = The number of the MCH specifier, used by the DMD and INS specifiers to reference the MCH.

GD_NUM = Reference number of a GDP specifier used to display all or part of the machine information for a disassembly line.

Default:

No default. This specifier must be entered by the user.

GDP

Description:

Each GDP (Group Descriptor) specifier describes a changeable field in the disassembly display line. Each specifier picks a sample word from the instruction sample array and isolates a field from that word in the same manner as the OPC specifier. The resulting field is "operated" on by the rest of the elements of GDP.

Format:

GDP (REF, IDX, MSK, VO_NUM, DSPLY_TYP, DT_PAR, GDL_NUM, OB_IDX)

where:

REF = The number of the GDP specifier, used by the INS and MCH to reference the GDP specifier.

IDX = Index into instruction sample array of word containing the field to be masked. The following code must be added to the IDX to determine the data group to be referenced:

00H - DATA group.
20H - STATUS group.
40H - ADDRESS group.
80H - MISC group.

MSK = Double word. Contains the mask for the group index field in instruction. A one in any bit location signifies that the corresponding bit in the instruction sample is to be used as part of the group index field. Mask data are right-justified and separate mask groups are combined (see text). The total number of bits set may not exceed 16 if the masked data are to be used as an index in a DSPLY_TYP or if it is to be used in conjunction with VO_NUM.

VO_NUM = Reference number of the VLD array containing the valid opcodes for this field, or "ALL VALID" if all codes are valid.

DSPLY_TYP = How to display the group field. List follows:

0 = Binary display of field.

1 = Octal display of field. Number of digits used is determined by the number of bits set in MSK.

2 = Hexadecimal display of field. Number of digits used is determined by the number of bits set in MSK.

3 = Mnemonic display list. This type requires a GDL array. The bit combination in the field determined by MSK is used as an index into the array.

4 = Signed hexadecimal display of field. Number of digits used is determined by number of bits set in MSK. MSB of field is used as sign bit. DT_PAR is used as an offset in the display.

5 = Bit map display. This type requires a GDL array. Each bit position in the field determined by MSK is assigned to an entry in the GDL array. If the bit is set in the sample word, that entry is displayed, otherwise, the next bit position is tested.

6 = Return read sample count of instruction. This type requires a GDL array. The bit combination in the field determined by MSK and IDX is used as an index into the array. Each element in the array contains a count of the number of samples each instruction the field causes to be read.

7 = Return sample count of instruction. This type requires a GDL array. The bit combination in the field determined by MSK and IDX is used as an index into the array. Each element in the array contains a count of the number of samples each instruction in the field requires

DT_PAR = Word. A general purpose variable used to pass a parameter to the DSPLY_TYP chosen if required. The keyword "NULL" may be used if this parameter is not needed.

GDL_NUM = Number of the GDL array containing display data used by several of the display types. The keyword "NULL" may be used if this parameter is not used.

OB_IDX = Starting location of display in output buffer, modulo 128. If bit seven is set, display is second line of a two-line display.

Default:

No default. This specifier must be entered by the user.

VLD

Description:

VLD (Valid) is a numbered, word array of valid bit combinations for the field designated by the OPC specifier referencing the array. Each entry is represented in a right-justified manner; i.e., an actual bit pattern of 11010000B should be entered as 00001101B.

Format:

VLD (REF, CODE_0 [, CODE_1, ... CODE_n])

where:

REF = The number of the VLD array, used by the OPC and GDP specifiers to reference the array.

CODE_xx = Word. A bit pattern valid for the field, right-justified.

Default:

All bit patterns valid. The default is specified by the user setting VO_NUM to OFFFFH (ALLVALID) in the OPC or GDP specifier.

OCL

Description:

OCL (Opcode Branch List) is a numbered, word array of references to OPC specifiers for each bit combination of the field designated by the OPC specifier referencing the array. This list provides the means of branching from one level to the next in the OPC tree.

Format:

OCL (REF, NUM_0 [, NUM_1 ... NUM_n])

where:

REF = The number of the OCL array used by the OPC specifier to reference the array.

NUM_xx = An OPC specifier reference number.

Default:

No default, may be specified by user, if needed.

TPL

Description:

TPL (Template) is a numbered ASCII array containing the static portions of a display line for a particular instruction or group of instructions.

Format:

TPL (REF, OB_IDX, string)

where:

REF = The number of the TPL array, used by the INS specifier to reference the array.

OB_IDX = Location on the display line of template string; range is 0 to 43.

string. An array of up to 44 characters, enclosed by single quotes.

Default:

No default, must be specified by user.

GDL

Description:

GDL (Group Display List) is a numbered array containing the data used by several of the display types from the GDP specifier. This array is generally an array of arrays, each sub-array containing an instruction mnemonic. It may also be used to pass information about the length of instructions in the INS group referencing the GDP specifier.

Format:

GDL (REF, [LEN], array)

where:

REF = The number of the GDL array, used by the GDP specifier to reference the array.

LEN = Optional. Length of each element of array in bytes. (Not used when passing length of instructions.) Range is 0 to 255.

array = An array of byte data, may be an array of characters enclosed by single quotes.

Default:

No default, may be specified by user, if needed.

UDD UUTIL OPERATION

GENERAL

The editor is part of a K205 utility program file labeled UUTIL.EXE. The file is loaded using the K205 DOS '.EXE' file load procedure. No other utilities may be loaded when the UUTIL is loaded.

UUTIL consists of a line-oriented editor and several utility commands to support the editor. The editor supports an RS-232 dumb terminal for database entry. The terminal is installed at the Auxiliary port of the K205. The Set-Terminal (SETTERM) command allows the user to configure the editor to the type of terminal in use. Refer to Appendix J for information about installation of a terminal.

Commands

When the UUTIL is loaded, the CRT is cleared and a directory of the current disk drive is displayed. Beneath the directory display is the UUTIL command line. The UUTIL Quick-key commands are listed in Table 6-1.

Table 6-1. UUTIL Quick-Key Commands

KEY	COMMAND
0	EDIT - CP/M 'ED'-type editor, incremental version output filename.
1	KLIST - List the contents of a file on the K205 screen. If a line is longer than 52 characters, then the rest of the line is truncated.
2	TLIST - List the contents of a file on the terminal.
3	LOCK - Write-protect a file using software routine.
4	UNLOCK - Remove write protection of a file using software routine.
5	SETTERM - Setup the codes expected by the K205 editor from the terminal or sent to the terminal.
6	DOWNLINK - Down loads a file from another host, formats the file to be understood by UDDGEN, and saves the file on disk.
7	EXIT - Exit UUTIL. Returns to K205 FORMAT screen.

Filename Entry

Most UUTIL utility commands require the user to enter a filename. There are two methods of filename entry available, both of which are identical to their counterparts in the K205 Disk Operating System.

- Auto-directory Method
- Edit-mode Method

Auto-Directory Method. The auto-directory method is the default method. This method requires that the desired filename exist in the directory of the disks in either drive. The filename is selected by manipulating the command cursor which is a reverse-video block appearing in the command fields. To select a file, perform the following sequence of steps:

1. Select the appropriate disk directory; press the F3 key successively until the correct directory is displayed on the Logic Analyzer display screen.

2. Move the command cursor from the command field to the filename field in the right of the command line; press the cursor-right key until the cursor appears in the filename field.

3. Move the filename into the filename command field; press the cursor-up or cursor-down keys, until the desired filename, from the directory, appears in the filename command field.

   A filename cursor also moves in the directory as the command cursor entry changes. (Quick keys 0 to F provide direct selection of the first 16 filenames in the directory, by position.)

The filename can be modified by entering the EDIT mode, using the cursor arrow keys to position the cursor, and the ASCII data keys to change the contents of the filename block.

EDIT-Mode Method. The second method is EDIT-mode entry. While in the filename field of the command line, press the EDIT key. The command cursor blinks. Using the left-cursor or right-cursor keys, move the cursor to either filename field and make entries using the ASCII data keys.

Filenames must not contain leading or imbedded spaces, and may not be blank.

K205 Function Keys

The functions of the K205 Function keys, when in UUTIL command-line mode, are listed in Table 6-2.

Table 6-2. UUTIL Command-Line Function Keys

KEY	FUNCTION
F1	Increment filename version (INC VER) when in EDIT mode or when auto-directory cursor is not on filename.
F2	Decrement filename version (DEC VER) when in EDIT mode or when auto-directory cursor is not on filename.
F3	Erase current directory (DIR), and replace by the directory of the opposite disk drive. If the opposite drive does not contain a disk, the current directory is redisplayed.
F4	Begin execution (EXECUTE) of current UUTIL command.
HELP	Print a help message appropriate to the current command-line field and mode.

UTILITY COMMANDS

EDIT Command

THE EDIT utility command is used to select the editor by pressing quick key 0, or by using the NEXT or PREV keys to step through the UUTIL command set. The UUTIL command line displays the filename of the file to be edited.

The displayed filename is the input file to the editor. The input file may be an existing editor file, or a filename not residing in the disk directory.

EDIT automatically increments the version number of the corresponding output filename after editing.

After selecting the desired filename, press the F4 key to begin editing. When in the EDIT mode, control is transferred to the terminal. The second line of the K205 CRT displays a message indicating the start and end of editing.

To abort EDIT and return to UUTIL command mode at any time during editing, press the F3 key. All editing changes are ignored.

Terminal commands available to the user when in the EDIT mode are listed in the next section, Editor Operations.

KLIST Utility Command

The KLIST utility command is used to examine editor or list files on the display screen of the K205. File lines longer than the 52-character width of the Logic Analyzer's CRT are truncated for display.

Select KLIST by pressing quick key 1, or by using the NEXT or PREV keys to step through the UUTIL command set. The UUTIL command line displays the single filename of the file to be listed.

After selecting the filename, press the F4 key to begin the list operation. The second line of the K205 CRT displays a message indicating the start and end of the listing. When in the KLIST mode, the function keys may be used to control the list operations. Table 6-3 contains the KLIST F-Key functions.

Table 6-3. KLIST F-Key Functions

KEY	FUNCTION
F1	Suspend (PAUSE) the file list operation
F2	Continue the file list operation
F3	Abort the file list operation and return to the UUTIL command mode
F4	(Blank)

When the listing ends, the last twenty lines of the file remain displayed on the K205 CRT until the F3 key is pressed.

TLIST Utility Command

The TLIST utility command is used to examine editor or list files on the terminal display screen attached to the auxiliary port of the K205.

Select TLIST by pressing quick key 2, or by using the NEXT or PREV keys to step through the UUTIL command set. The UUTIL command line displays a filename representing the file to be listed.

After selecting the filename, press the F4 key to begin the list operation. The second line of the K205 CRT displays a message indicating the start and end of the listing.

When in the TLIST mode, some control is transferred to the terminal keyboard. Table 6-4 contains the terminal control commands.

Table 6-4. Terminal Control Commands

KEYS	FUNCTION
Control/S	Suspend listing
Control/Q	Resume listing
Control/C or ESC	Abort listing and return to the UUTIL command mode
K205 F3 Key	Abort listing and return to the UUTIL command mode

When the listing ends, control is returned to the UUTIL command-line mode.

LOCK and UNLOCK Utility Commands

The LOCK command write-protects a file using a software routine rather than mechanical means; the UNLOCK command reverses the function.

LOCK and UNLOCK are selected by pressing quick key 3 or 4, respectively, or by using the NEXT or PREV keys to step through the UUTIL command set. The function of these commands is identical to the K205 DOS commands of the same name.

After selecting a filename, press the F4 key to execute the command.

SETTERM Utility Command

SETTERM is used to configure the editor to match a specific terminal. SETTERM determines the codes expected by the Logic Analyzer's editor when a control function key is pressed. The codes are sent to the terminal in response to a command.

Select SETTERM by pressing quick key 5, or by using the NEXT or PREV keys to step through the UUTIL command set. The UUTIL command line displays a filename representing the file containing the terminal characteristics. An existing SETTERM file may be edited using the SETTERM command.

After selecting the filename, press the F4 key to begin the operation. The second line of the K205 CRT displays a message indicating the start and end of the listing.

Lines 5 to 25 of the K205 CRT are blanked out and replaced by a matrix of key command codes. To enter the correct key-code sequence, use the cursor-right and cursor-left keys to move to the desired key-code position.

Each code sequence allows up to four eight-bit hexadecimal codes to be specified. If less than four codes are required for a particular function, the remaining entries must be set to zero. If a terminal does not support a function, all the entries for that function must be set to zero.

When all entries are complete, press the F4 key to write the resulting table data to disk under the selected filename. Press the F3 key to enter the displayed codes into the editor system.

The default terminal for the SETTERM command is the Lear Siegler ADM-3A terminal.

Utility Command Codes. The editor-generated and editor-expected codes are listed in Table 6-5.

Table 6-5. Utility Command Codes

TERMINAL CURSOR-KEY OUTPUT CODES	
CODE	COMMAND
CH	Cursor home
CU	Cursor up
CD	Cursor down
CL	Cursor left
CR	Cursor right

EDITOR GENERATED CURSOR-MOVEMENT CODES SENT TO THE CRT	
CODE	COMMAND
MH	Move cursor home
MU	Move cursor up
MD	Move cursor down
ML	Move cursor left
MR	Move cursor right
ES	Erase screen code
ER	Erase rest of screen code
EK	Erase line code
EL	Erase rest of line code
BK	Blankout code
AB	The BREAK character code

DOWNLINK Utility Command

The DOWNLINK command loads a file from another computer system, processes the file to a format that can be understood by UDDGEN, and automatically saves the file on disk.

The external computer system must be capable of sending ASCII-character files. When using the external computer to generate a data file for DOWNLINK, the user must put a <Control Z> character at the end of the text file. Otherwise, the DOWNLINK procedure does not recognize the end of the file.

Use RS-232 to communicate between the host machine and the Logic Analyzer. The communication protocol is XON/XOFF. See Appendix J for setup of auxiliary port.

To operate the DOWNLINK utility, perform the following sequence of steps:

1. Load UUTIL-xx.EXE file

2. When the directory screen is displayed, use NEXT/PREV key or "6" quick key to select the DOWNLINK command.

3. Select the appropriate filename.

4. Press the F4 key to execute the command. The K205 CRT screen is cleared and the message "Ready for downlink..." appears.

5. Send the file from the external computer to the K205. As the file is being sent through the RS-232, the K205 CRT display flashes the message, "Downlink ...", indicating that the Logic Analyzer is receiving the file.

6. When DOWNLINK is finished, the message "command done" appears at the top of the K205 CRT screen, and new directory information is displayed.
7. During DOWNLINK, press the F3 key to abort the process.

NOTE: The user must execute the DOWNLINK command before sending a file to the Logic Analyzer.

EXIT Utility Command

The EXIT utility command is used to leave the UUTIL program. EXIT saves all files in process, and transfers control of the Logic Analyzer to the K205 Setup screen.

Select EXIT by pressing quick key 7, or by using the NEXT or PREV keys to step through the UUTIL command set.

The UDD Editor

The UDD editor is a line-oriented editor fashioned after the CP/M 'ED' editor. Its command set is a subset of that editor, with a few minor differences in operation of some of the commands.

The editor reads text from the source file and places it in a memory buffer (APPEND). While in the memory buffer, the text may be viewed or altered using the set of line-editor commands described in this section. Edited text is written to a temporary output file to make room for additional source text using the write command. The memory buffer can hold up to 10240 characters. Currently, the maximum size of a text file is 65536 characters.

Upon receipt of an exit command, the editor writes the text, in the memory buffer, to the temporary file and then copies any remaining text in the source file to the temporary file. The temporary file is assigned the output filename and control returned to the K205 UUTIL command line.

Unlike the exit command, the quit command deletes the temporary file, closes the source file, and returns control to the K205.

The structure of the UUD Editor files system is illustrated in Figure 6-1.

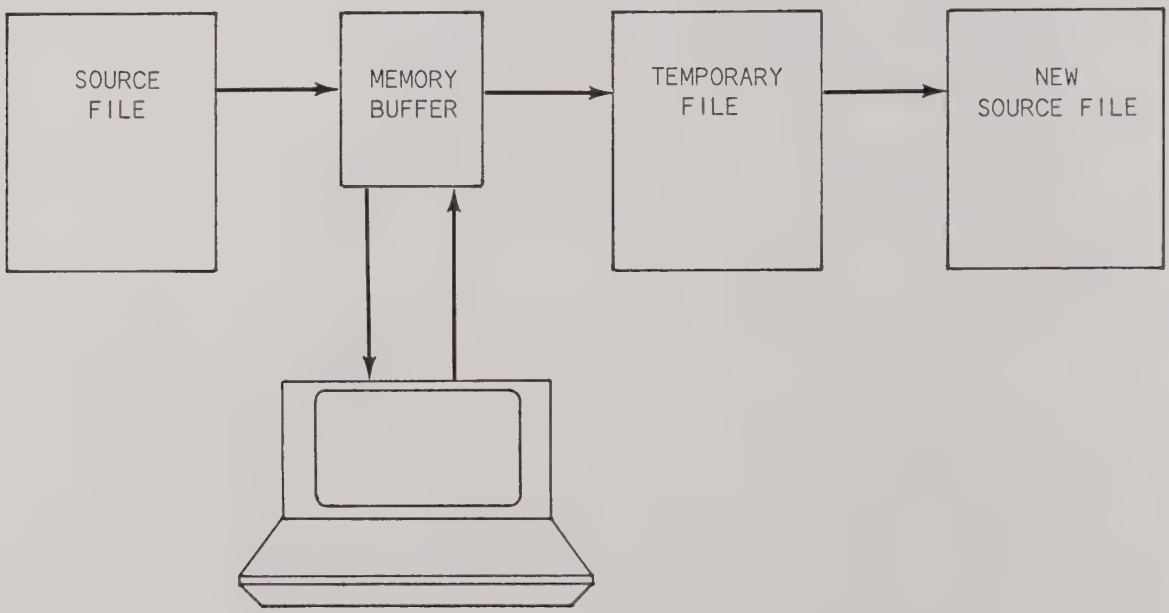


Figure 6-1. UDD Editor File System

Editor Text Pointer

Throughout this section of the manual, 'CP' refers to the position of the editor's character pointer. Note that the CP never points at a character, it is always 'between' characters, and the operation being performed operates on characters before or after the CP.

Upper/lower Case Characters

In the command lines, the editor treats upper and lower case characters the same. All commands may be specified by upper or lower case characters.

Line Editor

Within the editor is a smaller editor called the "line editor." This editor allows the user to modify a command or insertion-line before it is submitted to the text editor itself. The control characters used for editing in the line editor are listed in Table 6-6.

Table 6-6. Line-Editor Control Characters

KEYS	FUNCTION
<DEL>	Delete the character to the left of the CP in the line buffer. Will not delete past begining of line.
^ R	Retype the current command or insertion line.
^ E	Output a Carriage Return and Line Feed to the terminal, but does not affect command or insert line.
^ Z	String terminator in I, S, and F commands. Also, can be used as a command terminator for other commands, but is not necessary.
^ C	Immediate Quit command without Y/N prompt.
^ X	Start-over command line.

Edit Commands

The editor supports 16 commands. All of the commands that deal with text may be prefaced with a repetition value, n, as a positive number, negative number, or zero. If a value is not specified, n is assumed to be positive 1.

As a convenience, the symbol '#' may be specified in place of the repetition value number. The editor will interpret this as positive 65535, the maximum size of an entire file. For efficiency, most commands may be entered together on a single line.

The edit commands described in this section are listed in Table 6-7.

Table 6-7. UDD Edit Commands

COMMAND	MNEMONIC	FUNCTION
A	APPEND	Append a line of text to memory buffer
B	BEGINNING	Move to the Begining of memory buffer
C	CHARACTERS	Move CP by Characters
D	DELETE	Delete characters
E	EXIT	Exit editor
F	FIND	Find a string
H	HEAD	Move to Head of new file
I	INSERT	Insert new text
K	KILL	Delete (Kill) a line
L	LINES	Advance CP by Lines
O	ORIGINAL	Return to Original file
Q	QUIT	Quit editor
S	SUBSTITUTE	Substitute a string
T	TEXT	Display Text
V	VISUAL	Enter Visual line number mode
W	WRITE	Write a line to temporary disk file

Commands that require specific termination are noted as such in this section. The symbol '<CR>' indicates the RETURN key of the terminal.

nA<CR>

The APPEND command copies the next n unused source lines from the source file to the end of the memory buffer. Negative numbers are converted to absolute positive numbers.

±B<CR>

The BEGINNING command moves the CP to the beginning of the memory buffer if positive, or the end of the memory buffer if negative.

±nC<CR>

The CHARACTERS command moves CP by ±n characters, counting the end of a line as one character.

±nD<CR>

The DELETE command deletes n characters following the CP if positive, or deletes n characters before the CP if negative.

E<CR>

The EXIT command ends the current editing session. EXIT copies all text in the memory buffer to a temporary file, and copies any remaining text from the source file to the temporary file. The source file is then closed and the temporary file is renamed to the appropriate output filename. Control is returned to the K205 UUTIL command line.

This command must be issued singly and be terminated by a <CR>.

±nF<string>[^Z ¦ <CR>]

The FIND command searches the memory buffer for n text strings matching <string>. The search is conducted line by line on text following the CP if positive, before the CP if negative. <string> may not contain a <CR> character. If the search is successful, the CP is positioned before the first character of the last match found in the memory buffer. The number of matches found is reported on the terminal.

Since the CP is left positioned before the FIRST character of the target string, the CP is temporarily incremented by one before a search is attempted. This is so that multiple searches for the same string may be performed manually by the user without the need for advancing the CP past the last found string. As a result of this feature, it is necessary to backup the CP manually prior to a Find command if it is pointing to a string that matches the search string.

Control/Z may be used to terminate the Find command when it is to be used in a multiple command line. The maximum length of the Find string is 60 characters.

H<CR>

The HEAD command moves to the Head of new file. This command is equivalent to performing an Exit command and reinvoking the editor using the output filename of the previous edit command as the source filename. The memory buffer is emptied and the user must issue a new append command to load the buffer from the source file.

This command must be issued singly and terminated by a <CR>.

I<string>[^Z | <CR>]
I<CR>

The INSERT command inserts new text before CP. There are two forms of the insert command. The first form inserts a single string of text into the memory buffer. The string is entered on the command line with the Insert command. If it is terminated by a control/Z, the string becomes part of the current line. If the string is terminated by a <CR>, a <CR> is entered into the memory buffer and any text remaining on the current line becomes a new line in the memory buffer.

The second form of the insert command inserts multiple lines into the memory buffer, and is terminated by control/Z. If the insert was started in the middle of a line, the first portion of the line is printed on the terminal prior to insertion. The command line edit functions <DEL> and control/R also affect this portion of the current line. The line edit function <DEL> will not delete past the beginning of a line.

±nK<CR>

The KILL command deletes n lines following the CP if positive or before the CP if negative. If the CP is in the middle of a line, only that portion before or after the CP is affected, and that portion counts as one line in the repetition count. KILL will not delete past the end of the memory buffer. Text remaining in the source file must be appended to the memory buffer before it can be deleted.

±nL<CR>

The LINES command moves the CP forward (+) or backwards (-) n lines. If n = 0, then LINES moves the CP to the beginning of the current line. The CP will stop at the bottom or top of the memory buffer, respectively, if n is larger than the amount of lines between the CP and the bottom or top of the buffer.

O<CR>

The ORIGINAL command returns to Original file. This command is equivalent to performing a Quit command and reinvoking the editor using the original source filename. The memory buffer is emptied, the temporary file is deleted, and the user must issue a new append command to load the buffer from the source file. The effect of this command is to cancel all editing done in the current session and start over.

This command must be issued singly and be terminated by a <CR>.

Q<CR>

The QUIT command quits the edit session with no alterations to file. No output file is generated and control is returned to the UUTIL utility command line.

This command must be issued singly and be terminated by a <CR>. A 'Y' or 'N' is expected in response to the question "Quit editing - Y or N:".

±nS<string_1>^Z<string_2>[^Z | <CR>]

The SUBSTITUTE command performs a string search identical to the Find command, using <string_1> as the search string. If a match is found, <string_1> is replaced with <string_2> in the memory buffer. <string_1> is separated from <string_2> by a control/Z and the entire command is terminated with either a control/Z or a <CR>. Neither the control/Z or <CR> is included in the search or replacement strings. Upon successful completion of the command, the CP is left at the end of the last replacement string inserted and the number of substitutions made is reported on the terminal. The maximum length of the Find or Substitute strings is 60 characters. SUBSTITUTE does not work in a "text bufferfull" situation.

±nT<CR>

The TEXT command types the contents of the current line and n - 1 lines from the CP forward (+) or backward (-). If n = 0, it then type the contents of the current line up to the CP. The listing will stop at the bottom or top of the memory buffer, respectively, if n is larger than the amount of lines between the CP and the bottom or top of the buffer. Control/S and control/Q will cause the listing to pause and continue, respectively. The <ESC> key is used to abort long listings.

±V<CR>

The VISUAL command, if positive, causes the editor to enter the visual line number mode. Until a -V command is invoked, all command and insertion mode lines will display the line number that the CP is currently on. The line number reflects any lines written to the temporary file. If the CP is at the end of the memory buffer, the line number will be blank.

nW<CR>

The WRITE command writes the first n lines of the memory buffer to the temporary file, and moves the remaining lines in the memory buffer to the top of the buffer. This command is used when the memory buffer is full to allow entering or appending more text.

±n<CR>

±n is functionally equivalent to the command string +/-nLT, moving the CP forward or backward n lines and typing the final line.

:

The editor allows absolute line number operations through the use of the colon ':'. Appending a colon to an absolute line number will move the CP to the beginning of that line.

23:

23: moves the CP to line 23 if present in memory buffer. Otherwise, the CP moves to the nearest line number. Prepending a command with a colon followed by an absolute line number will cause that command to operate up to, and including, the specified line.

23::25T

23::25T moves the CP to line 23 and the next three lines, up to line 25, to be typed.

UDD PREPROCESSOR OPERATION

GENERAL

The Preprocessor is part of a K205 utility program file labeled UDDGEN.EXE. The file is loaded using the K205 DOS '.EXE' file load procedure. No other utilities may be loaded once UDDGEN is loaded.

UDDGEN consists of the preprocessor and several list utility commands.

Commands

When UDDGEN is loaded, the CRT is cleared and a directory of the current disk drive is displayed. Beneath the directory display is the UDDGEN command line. The UDDGEN preprocessor commands and their associated quick-key numbers are listed in Table 7-1.

Table 7-1. UDDGEN Preprocessor Quick-Key Commands

KEY	COMMAND
0	PREPROC - Select the UDDGEN preprocessor program.
1	KLIST - List the contents of a file on the K205 screen. A line longer than 52 characters is truncated.
2	TLIST - List the contents of a file on the terminal.
3	EXIT - Exit UUDGEN. Returns to K205 FORMAT screen.

The user can enter the desired file name by two methods:

- Auto-Directory Method
- EDIT-Mode Entry Method

Auto-Directory Method. The auto-directory method is the default method. This method requires that the desired filename exist in the directory of the disks in either drive. The filename is selected by manipulating the command cursor which is a reverse-video block appearing in the command fields. To select a file, the user performs the following sequence of steps:

1. Select the appropriate disk directory; press the F3 key successively until the correct directory is displayed on the K205 CRT.

2. Move the command cursor from the command field to the filename field in the right of the command line; press the cursor-right key until the cursor appears in the filename field.

3. Move the filename into the filename command field; press the cursor-up or cursor-down keys, until the desired filename, from the directory, appears in the filename command field. A filename cursor also moves in the directory as the command cursor entry changes.

The filename can be modified by entering the EDIT mode, using the cursor arrow keys to position the cursor, and the ASCII data keys to change the contents of the filename command field.

EDIT-Mode Entry. The second method is EDIT mode entry. While in the command-name field of the command line, press the EDIT key. The command cursor starts to blink. Using the left or right cursor keys, move the cursor to either filename field and begin entry using the ASCII data keys.

Filenames must not contain leading or imbedded spaces, and may not be blank.

Begin operations

After selecting a file name, press the F4 key to begin preprocessor operation. CRT line two displays messages indicating the beginning and end of processing. Once processing has begun, the F3 key may be used to abort the operation and return to UDDGEN command mode.

PREPROCESSOR INPUT-FILE CONSTRUCTION

The UDDGEN utility command is used to generate database files for the UDD from source files. The UDDGEN PREPROCESSOR command is selected by pressing quick key 0, or using the NEXT or PREV keys to step through the UDDGEN command set. The UDDGEN command line then displays a filename. The filename is the preprocessor input file, which is generated by the the UUTIL utility.

The preprocessor input file is generated using the UUTIL utility supplied with the UDD or by an external editor compatible with the UUTIL. Externally generated files may be loaded into the Logic Analyzer through the separate DOWNLINK utility.

The UDD programmer defines the format of UDD through use of the preprocessor. Entries are made in CRT display lines 5 to 25 and columns 8 to 51 inclusive. Each target processor instruction may be displayed on one or two CRT lines. The output buffer, referred in the following section, is an analog of the user defined portion of the UDD display line, indexed from 0 to 43. In actual use the user has available two buffers designated OB_1 and OB_2, describing the first and second lines of an instruction, respectively.

PREPROCESSOR OUTPUT FILES

The preprocessor produces two kinds of output files:

- Database-Object File
- List File

Database-Object File

The database-object file is a standard DOS utility file containing the database object code produced by the preprocessor. The source filename is used with an extension of '.EXE' for the output object filename. The preprocessor control '$NOOBJECT' may be inserted into the source file prior to any database statements to prevent generation of the object file. An object file is not generated if errors are present in the source file.

List File

The list file contains the source text, line numbers, database information, and any error messages. The list file is the same DOS type as the preprocessor source file. The source filename is used with an extension of '.LST' for the list filename. The preprocessor control '$NOPRINT' may be inserted into the source file prior to any database statements to prevent generation of the list file. If the $NOPRINT control is in force, error messages will be output to the logic analyzer CRT screen.

Output-File Controls

The preprocessor controls '$LIST' and '$NOLIST' may be interspersed in the source text to control what source lines are transferred to the list file. A '$LIST' control is assumed at the beginning of the source file. Source lines containing errors will be listed regardless of the current list control.

All preprocessor controls ('$xxxxxx') must begin in the left-most column of a source text line.

LIST-FILE INFORMATION

The preprocessor list file contains the following information located at the end of the list file:

- Names of the database statements for which the default was used.
- Names of source, object, and list files.
- Number of source file lines read.
- Number of warnings issued.
- Number of errors detected.
- Size of object file in bytes.
- Size of free database memory.

KLIST UTILITY COMMAND

The KLIST utility command is used to examine editor or list files on the display screen of the K205. File lines longer than the 52-character width of the Logic Analyzer's CRT are truncated for display.

Select KLIST by pressing quick key 1, or by using the NEXT or PREV keys to step through the UUTIL command set. The UUTIL command line displays the single filename of the file to be listed.

After selecting the filename, press the F4 key to begin the list operation. The second line of the K205 CRT displays a message indicating the start and end of the listing. When in the KLIST mode, the function keys may be used to control the list operations. Table 7-2 contains the KLIST F-Key functions.

Table 7-2. KLIST F-Key Functions

KEY	FUNCTION
F1	Suspend (PAUSE) the file list operation
F2	Continue the file list operation
F3	Abort the file list operation and return to the UUTIL command mode
F4	(Blank)

When the listing ends, the last twenty lines of the file remain displayed on the K205 CRT until the F3 key is pressed.

TLIST UTILITY COMMAND

The TLIST utility command is used to examine editor or list files on the terminal display screen attached to the auxiliary port of the K205.

Select TLIST by pressing quick key 2, or by using the NEXT or PREV keys to step through the UUTIL command set. The UUTIL command line displays a filename representing the file to be listed.

After selecting the filename, press the F4 key to begin the list operation. The second line of the K205 CRT displays a message indicating the start and end of the listing.

When in the TLIST mode, some control is transferred to the terminal keyboard. Table 7-3 contains the terminal control commands.

Table 7-3. Terminal Control Commands

KEYS	FUNCTION
Control/S	Suspend listing
Control/Q	Resume listing
Control/C or ESC	Abort listing and return to the UUTIL command mode
K205 F3 Key	Abort listing and return to the UUTIL command mode

When the listing ends, control is returned to the UUTIL command-line mode.

EXIT UTILITY COMMAND

The EXIT utility command is used to leave the UUTIL program. EXIT saves all files in process, and transfers control of the Logic Analyzer to the K205 Setup screen.

Select EXIT by pressing quick key 7, or by using the NEXT or PREV keys to step through the UUTIL command set.

UDD DISASSEMBLER OPERATION

GENERAL

Connections

Connect the Logic Analyzer to the target system according to the recommendations in the K205 Logic Analyzer User's Manual. It is the user's responsibility to prepare the interface. The UDD-205 program resides on the UDD diskette. Place the UDD-205 diskette into drive A of the K205 Logic Analyzer, and any additional diskette containing the database into drive B.

Load the UDD

Load the UDD using the Logic Analyzer DOS '.EXE' file load procedure. No other utilities may be loaded while the UDD is installed. Access the UDD by pressing Quick-Key 4 on the DATA FORMAT screen of the Logic Analyzer; this selects the disassembler mode. Thereafter, entering the DATA display screen displays disassembled data rather than the normal DATA screen format.

After loading the UDD, the user loads the appropriate database file. This file also has an extension of '.EXE', but is not an executable file. The file is produced by the UDDGEN Preprocessor command and contains the database for the customized mnemonic display.

The sequence of steps for loading the UDD and datafile is summarized as follows:

1. Place the AC POWER switch to the ON position to power up the K205.

2. Gently insert the UDD-205 diskette into Drive A with the disk slot toward the rear of the unit and the label up. Lock the disk in place using the drive latch handle.

3. Similarly, insert the data diskette into drive B.

4. Press I/O and numeric 1 keys to display the Disk Screen Mode (DOS).

5. Press the F3 key to display directory B.

6. Press Quick Key 1 to select recall function.

7. Using cursor keys, select the UDD205.EXE file for loading. Press function key F4 to execute loading. A HELP message appears. Press any key to eliminate the message; a file directory appears.

8. Using cursor keys, select the desired datafile for loading. Press function key F4 to execute loading.

9. Press the FORMAT and numeric key 4 to select the disassembler data format screen.

10. Press the DATA key to start disassembling.

```
* * * * * * * * * * * * * * * * * * * *
*                CAUTION                *
*                                       *
*  Make a backup copy of the master     *
*  disk prior to using the UDD-205.     *
*                                       *
* * * * * * * * * * * * * * * * * * * *
```

RECORD SETUP

Record parameters for thresholds, polarity, and clocking are loaded automatically by the UDD. The user can modify the user-defined format to suit specific needs without affecting the UDD.

Record parameters can be altered to suit a particular need, and saved in a '.SM' file for future recall. A backup should be made of the UDD205.EXE and data files before using.

If the UDD is loaded and active (selected on the FORMAT screen) and another disassembler, UDD or standard, is loaded, the user must return to the FORMAT screen and reselect the disassembler format for correct operation.

Key processing for the Logic Analyzer record parameters is handled in the same way when the UDD is loaded with the exception of the trace control pattern fields.

KEYBOARD OPERATION

Display Operations

The UDD display screen is a modified version of the Logic Analyzer DATA display screen. UDD cursor movement is slightly different.

The UDD displays some instructions on two successive CRT lines. When scrolling, the cursor operates as if the two lines were a single line, always moving to the first of the two lines. When scrolling to the first line of a two-line instruction, an extra scroll is performed to move the second line onto the screen.

Because multi-byte instructions require more than one trace sample location, cursor movement is not entirely linear. In all cases, the cursor moves up or down an entire assembly code line. When using the SET CURSOR function, the cursor moves to the sample number that begins an assembly code line. If a sample number is entered that is not at the beginning of an assembly code line, the cursor sample number is automatically adjusted to the beginning of the code line within which the entered number falls.

Display Format, Mnemonic/Object Format

To display records in object format, press the F3 soft key. Disassembled code from the control cursor forward is displayed in object format. Press soft key F4 to redisplay the data in disassembled mnemonic form. This feature is useful for realigning the disassembly process to recover from incorrect record data.

The UDD display screen soft key functions are summarized in Table 8-1.

Table 8-1. Disassembler Soft key Functions

KEY	FUNCTION
1	Page up
2	Page down
3	Object Format display
4	Disassemble mnemonic display

Display Format, Shift/S Set Mode

The UDD Set Mode provides a method of disabling display of certain portions of recorded data. When designing the database, the user makes provisions to inhibit the display of unwanted types of data. The Shift/S command selects the Set mode, and temporarily redefines the function keys.

To select the Set mode, press the SHIFT key and "S" key in succession. Then, to select the desired screen format, press the F1, F2, or F3 keys. Press function key F4 to display all recorded cycles. The selected partial-display format is indicated in the upper left-hand corner of the CRT.

When a screen format is selected, the UDD leaves the Set mode. To leave the Set mode without altering the display format, press any key other than soft keys F1 to F4. The format setting is maintained until one of the following conditions occurs:

- Selecting a new setting
- Initiating disassembly by pressing the F4 key
- Reloading the UDD from disk.

The UDD shift/S command soft key functions are summarized in Table 8-2.

Table 8-2. Set-Mode Soft key Functions

KEY	FUNCTION
1	Programmed by the user
2	
3	
4	Display all recorded samples

Attempting to initiate the SHIFT/S commands F1, F2, or F3 with no mnemonic code present in the display buffer results in the following error message on line two of the CRT:

"ONLY OBJECT CODE DISPLAYED."

This message is displayed until the user presses another key.

Disassembler Search Word

The UDD search feature allows the user to search recorded memory for the presence of a specific data pattern (assembly code), status code, or address. The search control line is displayed on CRT line 3, above the header line. The UDD search word is NOT the same as that entered in the main screen of the K205.

Press the NEXT key to move the active cursor to the next occurrence of the search word. Press the PREV key to move the cursor to the previous occurrence of the selected search word. If not found, the following message appears on CRT line 2 until another key is pressed:

"SEARCH WORD NOT FOUND"

UDD search is active when the search word contains symbols other than all "don't care" symbols. If the search word is not active, the NEXT and PREV keys locate the next or previous level change. If no level change is found, the cursor goes to the end or beginning of the record buffer.

The procedure for altering the UDD search word is similar to altering the search word of the Logic Analyzer DATA screen. The procedure consists of the following sequence of steps:

1. Press the EDIT key to enter the edit mode.

2. Move the cursor through the search-word field altering the search word as necessary.

3. Press the EDIT key a second time to leave the edit mode. The entry residing in the search-word field when leaving the edit mode becomes the active search word.

Trace-Control Pattern Selection

When the UDD is loaded and active, the selection method of the trace-control screen pattern is altered.

- Data-format screen labels replace the section labels.

- The status field displays a selection of status bit patterns pre-defined by the database designer for the processor.

The user can edit the status-field selection using the normal Logic Analyzer method. If the edited pattern matches one of the pre-selected patterns, that pattern name is displayed. Otherwise, the user-selected pattern is displayed.

While in the status field, the user can select the predefined patterns directly by pressing it's associated quick key, or sequentially by pressing the NEXT and PREV keys. The DON'T CARE (X) key selects the don't-care status pattern when in the status field.

DISASSEMBLER SCREEN DISPLAY.

This section describes the CRT display screen when in the normal display mode. Each line on the CRT display, in the range 5 through 25, contains a line of disassembled data or an object code.

Since the UDD can display disassembled instructions on two lines, if a second line is required, the second line will not contain the FRAME or LEVEL fields. The cursor does not stop on these lines.

Display lines are described in Table 8-3.

Table 8-3. CRT Display Lines

LINE	DESCRIPTION
1	The name of the disassembler. Used for various system messages such as "EDIT MODE".
2	Reserved for System and Disassembler error messages.
3	The search word.
4	The display header line. It contains the heading for each column in the display. Format is determined by the database designer.
5 - 25	Disassembler output.
27	The CONTROL and REF location fields. The time difference between the CONTROL and REF cursor positions is also displayed on this line.
28	Softkey definitions. The definition of the soft keys is dependent on the current mode of operation.
29	The System status line. The disassembler does not alter this line.
	All other lines are blank.

Information contained in the columns within lines 5 through 25 are described in Table 8-4.

Table 8-4. CRT Display Columns

COLUMN	DESCRIPTION
1	The cursor indicator. If the CONTROL or REFERENCE cursor is presently anywhere in the sample data associated with the source code line displayed, the character 'C' or 'R' is displayed. If both cursors are within the same data domain, the special double character 'Cr' is displayed. Otherwise, the column is left blank.
2 - 6	FRAME. Reserved for the first analyzer data sample number associated with the disassembly code for this line.
8 - 51	Disassembled data in the format determined by the database designer.
52	The Logic Analyzer level at which the sample was recorded.

DISASSEMBLER HELP FACILITY

The disassembler help facility is maintained separately from the main Logic Analyzer help program. When the user presses the HELP key, the contents of lines 5 through 25 are erased and a message appropriate to the situation is displayed along with a static display of disassembler commands.

Help messages are available for the following operations:

- Normal disassembly display
- Setting the display mode
- Editing the search word
- Setting the R or C cursors

To leave the help display, press any key. The previous display is restored.

DISASSEMBLER NOTES

Software Version Numbers, Shift/V

The UDD software version number and the custom database version number may be displayed on line two of the CRT by pressing the SHIFT and "V" keys in succession. The display will remain until another key is pressed.

Processing Indications

Disassembly processing is indicated by the following message flashing in the upper left-hand corner of the CRT:

"Disassembling"

If disassembled code is displayed on the CRT when the disassembly begins, it remains until the process is complete. Otherwise, the screen is cleared during the disassembly process.

SAMP Display

Although displayed, "SAMP" is never used as part of a disassembled instruction.

PREPROCESSOR EXAMPLE, STATE MACHINE

This appendix contains the UDD database description of a four bit code. It is used to illustrate the use of the UDD as a look-up table disassembler.

The following is a listing of the state machine.

CODE	NAME	CODE	NAME
0000	CURSOR UP	1000	ERASE TO END OF SCREEN
0001	CURSOR DOWN	1001	ERASE TO END OF LINE
0010	CURSOR LEFT	1010	DIRECT CURSOR ADDRESS
0011	CURSOR RIGHT	1011	IDENTIFY
0100	ENTER GRAPHICS MODE	1100	ENTER ALTERNATE KEYPAD
0101	EXIT GRAPHICS MODE	1101	EXIT ALTERNATE KEYPAD
0110	CURSOR TO HOME	1110	ENTER ANSI MODE
0111	REVERSE LINE FEED	1111	Not used (illegal).

The following pages contain the source listing for a table look-up database to decode the state machine.

```
    1   !
    2   tgp( DATA, AL)
    3   tgp( ADDRESS, AL)
    4   tgp( STATUS)
    5   tgp( MISC)
    6
    7   nam( 'State Decode')
    8   hdg( 'Object     Mnemonic')
    9
   10   hlp( ' ', ' ', 'This is a test of the ',
   11                  'emergency broadcast system.')
   12
   13
   14   !Here is the single opc used in this test.
   15
   16   opc( 0, 0, 0, ALLVALID, 0, NOTUSED, NOTUSED)
   17
   18   ins( 0, 0, 0,  0)     ! Looks empty, doesn't it (it's not!)
   19
   20   mch( 0, 1)    !Display the object code only
   21
   22   gdp( 0, 0, 0FH, ALLVALID, 3, 0, 0, 11)  !Print one of sixteen Mnemonics
   23   gdp( 1, 0, 0FH, ALLVALID, 1, 0, NULL, 0) !Print data in octal.
   24
   25   tpl( 0, 1, ' ')
   26
   27   gdl( 0, 22, 'Cursor Up              ', 'Cursor Down            ',
   28               'Cursor Left            ', 'Cursor Right           ',
   29               'Enter Graphics Mode    ', 'Exit Graphics Mode     ',
   30               'Cursor to Home         ', 'Reverse Line Feed      ',
   31               'Erase to End of Screen', 'Erase to end of line  ',
   32               'Direct Cursor Address ', 'Identify              ',
   33               'Enter Alternate Keypad', 'Exit Alternate Keypad ',
   34               'Enter ANSI Mode        ', '  -- Not Used --       ')
   35
   36
   37
   38   !  THE END!!!
   39
```



```
*** WARNING At THR          - default value used
*** WARNING At VAR          - default value used
*** WARNING At POL          - default value used
*** WARNING At CLK          - default value used
*** WARNING At VER          - default value used
*** WARNING At MSK          - default value used
*** WARNING At VTC          - default value used
*** WARNING At DMD          - default value used
*** WARNING At TCS          - default value used
*** WARNING At MMD          - default value used

Source file :           A:OTEST -10.DAT
Object file :           A:OTEST -10.EXE
List file   :           A:OTEST -10.LST
   39  Line(s) read
    0  Error(s) found
   10  Warning(s) issued
  485  Bytes used
14543  Bytes remaining
```

PREPROCESSOR EXAMPLE, 8085 MICROPROCESSOR

This appendix contains an example of a database for the UDD-205. It is based on the Intel 8085 Microprocessor.

```
  1   !Sample database for the Intel 8085 microprocessor.
  2   !
  3   !This sample program is intended as an aid to learning
  4   !to program the User Defined Disassembler database language.
  5   !As such, it is not guaranteed to be error free, nor is
  6   !the only, or even the best, method for implementing the
  7   !8085 disassembler with the UDD.
  8   !
  9   tgp( DATA, AL)
 10   tgp( ADDRESS, BL, AH)
 11   tgp( STATUS, CL)
 12   tgp( MISC, CL)
 13
 14   clk( 1, 20, 2, 1, 1, 1, 1,     ! External clock source
 15        1, 1, 1, 0, 0, 0,         ! Master clock expression
 16        0, 0, 0, 0, 0, 1,         ! A clock expression
 17        0, 0, 0, 0, 0, 1,         ! B clock expression
 18        0, 0, 0, 0, 0, 1,         ! C clock expression
 19        0, 0, 1, 0, 0, 0,         ! A enable expression
 20        0, 0, 0, 0, 0, 0,         ! B enable expression
 21        0, 0, 0, 0, 0, 0,         ! C enable expression
 22        1, 0, 0,                  ! Select demux on section A
 23        0, 1, 0, 0, 0, 0,         ! Latch section AH
 24        0, 0, 0)                  ! Sample all sections
 25
 26
 27   nam( '8085 Disassy')
 28
 29   hdg( 'ADDR OBJECT STATUS MNEMONIC      ')
 30
 31   hlp( ' ', ' ',
 32        'Shift/S Display modes:',
 33        '    F1 - Eliminate data read/write cycles',
 34        '    F2 - Mnemonic only display',
 35        '    F3 - Display all cycles',
 36        '    F4 - Display all cycles')
 37
 38   ! Status group inputs:
 39   !    D2      D1      D0
 40   !  IO/M-     S1-     S0-
 41
 42   msk( 07H)
 43   chk( 3, PWR, PRD, PRD, 3, 3, 3, 3)
 44
 45   mmd( 2, 0) ! 'OR' all MISC group instruction samples
 46
 47   vtc( 1, PF1, PF1, PF1, PF1, 3, NODATA)   !No data read/write cycles
 48   vtc( 2, PF1, PF1, PF1, PF1, PF1, NODATA) !Mnemonics only.
 49   vtc( 3, PF1, PRD, DRD, PWR, 3, NODATA)   !Display all cycles.
 50   vtc( 4, PF1, PRD, DRD, PWR, 3, NODATA)   !Display all cycles.
 51
 52   dmd( 0, 4, 4, 4, 4, 0)
 53
 54   tcs( 0, XXXXX010, 'READ    ')
 55   tcs( 1, XXXXX001, 'WRITE   ')
```

```
 56   tcs( 2, XXXXX110, 'INPUT   ')
 57   tcs( 3, XXXXX101, 'OUTPUT  ')
 58   tcs( 5, XXXXXX00, 'HALT    ')
 59   tcs( 4, XXXXX011, 'FETCH   ')
 60   tcs( 6, XXXXX111, 'ACK     ')
 61
 62   ! Here is the Opcode decoder tree.
 63
 64   opc( 0, 0, 0C0H, ALLVALID, NOTUSED, NOTUSED, 0)
 65   opc( 1, 0, 007H, ALLVALID, NOTUSED, NOTUSED, 1)        !00 xxx xxx ops
 66   opc( 2, 0, 03FH,        0,       0,       5, NOTUSED) !01 xxx xxx ops
 67   opc( 3, 0, 03FH, ALLVALID,       2, NOTUSED, NOTUSED) !10 xxx xxx ops
 68   opc( 4, 0, 007H, ALLVALID, NOTUSED, NOTUSED,       2) !11 xxx xxx ops
 69   opc( 5, 0, 03FH, ALLVALID,       1, NOTUSED, NOTUSED) !MOV
 70   opc( 6, 0, 038H, ALLVALID,      22, NOTUSED, NOTUSED) !cond. RET.
 71   opc( 7, 0, 038H, ALLVALID,       3, NOTUSED, NOTUSED) !pop and friends.
 72   opc( 8, 0, 038H, ALLVALID,       4, NOTUSED, NOTUSED) !cond. JUMPs
 73   opc( 9, 0, 038H,        1,       5,      14, NOTUSED) !JMP
 74   opc(10, 0, 038H, ALLVALID,       7, NOTUSED, NOTUSED) !cond. CALL.
 75   opc(11, 0, 038H,        3,       8,      16, NOTUSED) !CALL
 76   opc(12, 0, 038H, ALLVALID,      10, NOTUSED, NOTUSED) !imm math & logic
 77   opc(13, 0, 038H, ALLVALID,      11, NOTUSED, NOTUSED) !RSTx
 78   opc(14, 0, 038H,        2,       6,      15, NOTUSED) !IN & OUT
 79   opc(15, 0, 038H,        6,      23, NOTUSED, NOTUSED) !XTHL...
 80   opc(16, 0, 038H,        4,       9, NOTUSED, NOTUSED) !PUSH and friends
 81   opc(17, 0, 030H,        5,      12, NOTUSED, NOTUSED) !NOP and pals
 82   opc(18, 0, 008H, ALLVALID, NOTUSED, NOTUSED,       3) !LXI/DAD
 83   opc(19, 0, 020H, ALLVALID, NOTUSED, NOTUSED,       4) !STAX/SHLD
 84   opc(20, 0, 038H, ALLVALID,      17, NOTUSED, NOTUSED) !INX/DCX
 85   opc(21, 0, 038H, ALLVALID,      18, NOTUSED, NOTUSED) !INR
 86   opc(22, 0, 038H, ALLVALID,      19, NOTUSED, NOTUSED) !DCR
 87   opc(23, 0, 038H, ALLVALID,      20, NOTUSED, NOTUSED) !MVI
 88   opc(24, 0, 038H, ALLVALID,      21, NOTUSED, NOTUSED) !rotates
 89   opc(25, 0, 038H, ALLVALID,      13, NOTUSED, NOTUSED) !LXI
 90   opc(26, 0, 030H, ALLVALID,      14, NOTUSED, NOTUSED) !DAD
 91   opc(27, 0, 018H, ALLVALID,      15, NOTUSED, NOTUSED) !STAx
 92   opc(28, 0, 018H, ALLVALID,      16, NOTUSED, NOTUSED) !SHLD
 93
 94   ! Beginning of opc reference number lists.
 95
 96   ocl( 0, 1, 2, 3, 4)
 97   ocl( 1, 17, 18, 19, 20, 21, 22, 23, 24)
 98   ocl( 2, 6, 7, 8, 9, 10, 11, 12, 13)
 99   ocl( 3, 25, 26)
100   ocl( 4, 27, 28)
101
102   ! Beginning of Instruction blocks.
103
104   ins( 0, 0, 1)                ! HLT instruction
105   ins( 2, 2, 1,  1, 7, 35)     ! Arithmetic & logical
106   ins( 1, 1, 1,  0, 1, 28)     ! MOV instructions
107   ins(22, 2, 1,  8, 34)        ! conditional returns
108   ins( 3, 2, 1,  9, 33)        ! POP & friends
109   ins( 4, 2, 3,  10, 11, 12)   ! conditional jumps
110   ins( 5, 3, 3,  11, 12)       ! JMP
```

```
111   ins( 7, 2, 3,  14, 11, 12) ! conditional calls
112   ins( 8, 4, 3,  11, 12)     ! CALL
113   ins(10, 5, 2,  12, 16)     ! Immediate arith & logical
114   ins(11, 6, 1,  17)         ! RST x
115   ins( 6, 2, 2,  13, 12)     ! IN & OUT
116   ins(23, 2, 1,  24, 32)     ! XTHL...
117   ins( 9, 2, 1,  15)         ! PUSH & pals
118   ins(12, 2, 1,  18)         ! NOP....
119   ins(17, 2, 1,  22)         ! INX/DCX
120   ins(18, 8, 1,  0, 31)      ! INR
121   ins(19, 9, 1,  0, 31)      ! DCR
122   ins(20, 10, 2,  0, 12, 31) ! MVI
123   ins(21, 2, 1,  23)         ! rotates
124   ins(13,12, 3,  19, 36, 37) ! LXI...
125   ins(14, 7, 1,  19)         ! DAD
126   ins(15, 2, 1,  20, 29)     ! STAX...
127   ins(16, 2, 3,  21, 11, 12, 30) ! SHLD...
128
129   ! Beginning of Machine Blocks.
130
131   mch( 1,  2, 3, 27)      ! one byte instruction.
132   mch( 2,  2, 3, 4, 27)   ! two byte instruction.
133   mch( 3,  2, 3, 4, 5, 27)! three byte instruction.
134   mch( 0,  2, 3)          ! NODATA sample display.
135   mch( 4,  2, 6, 25, 26)  ! io/mem reads and writes and misc cycles.
136
137   ! Beginning of template
138
139   tpl( 0, 19, 'HLT')
140   tpl( 1, 19, 'MOV   ,')
141   tpl( 2, 19, '  ')
142   tpl( 3, 19, 'JMP')
143   tpl( 4, 19, 'CALL')
144   tpl( 5, 21, 'I')
145   tpl( 6, 19, 'RST')
146   tpl( 8, 19, 'INR')
147   tpl( 9, 19, 'DCR')
148   tpl(10, 19, 'MVI   ,')
149   tpl(12, 19, 'LXI    ,')
150   tpl( 7, 19, 'DAD')
151
152   ! Beginning of Valid Options lists.
153
154   vld( 0, 36H)                ! HLT
155   vld( 1,  0H)                ! JMP
156   vld( 3, 01H)                ! CALL
157   vld( 2, 02H, 03H)           ! OUT & IN
158   vld( 6, 04H, 05H, 06H, 07H)! XTHL...
159   vld( 4, 00H, 02H, 04H, 06H)! PUSH...
160   vld( 5, 00H, 02H, 03H)      ! NOP & pals
161
162   ! Beginning of Group Descriptors.
163
164   gdp(  0, 00H,    038H, ALLVALID, 3, NULL,    0, 24) !Dest register
165   gdp(  1, 00H,    007H, ALLVALID, 3, NULL,    0, 26) !Source register
```

```
166   gdp(  2, 40H,  OFFFFH, ALLVALID, 2, NULL, NULL,  0) !MCH address
167   gdp(  3, 00H,    0FFH, ALLVALID,  2, NULL, NULL, 5) !obj disp, byte 1
168   gdp(  4, 01H,    0FFH, ALLVALID, 2, NULL, NULL,  7) !obj disp, byte 2
169   gdp(  5, 02H,    0FFH, ALLVALID, 2, NULL, NULL,  9) !obj disp, byte 3
170   gdp(  6, 80H,    0F8H, ALLVALID, 5,    0,   16, 12) !MISC group disp
171   gdp(  7, 00H,    038H, ALLVALID, 3, NULL,    1, 19) !Arith & logical
172   gdp(  8, 00H,    038H, ALLVALID, 3, NULL,    2, 19) !cond. ret
173   gdp(  9, 00H,    038H, ALLVALID, 3, NULL,    3, 19) !stack ops
174   gdp( 10, 00H,    038H, ALLVALID, 3, NULL,    4, 19) !cond. jmp
175   gdp( 11, 02H,    0FFH, ALLVALID, 2, NULL, NULL, 24) !hi byte addr, mne
176   gdp( 12, 01H,    0FFH, ALLVALID, 2, NULL, NULL, 26) !lo byte addr, mne
177   gdp( 13, 00H,    008H, ALLVALID, 3, NULL,    5, 19) !in, out
178   gdp( 14, 00H,    038H, ALLVALID, 3, NULL,    6, 19) !cond. calls
179   gdp( 15, 00H,    030H, ALLVALID, 3, NULL,    7, 19) !double reg pushes
180   gdp( 16, 00H,    038H, ALLVALID, 3, NULL,    8, 19) !imm arith & logic
181   gdp( 17, 00H,    038H, ALLVALID, 3, NULL,   19, 24) !RST level
182   gdp( 18, 00H,    030H, ALLVALID, 3, NULL,    9, 19) !miscellaneous
183   gdp( 19, 00H,    030H, ALLVALID, 3, NULL,   10, 24) !Double reg names
184   gdp( 20, 00H,    018H, ALLVALID, 3, NULL,   11, 19) !indirect moves
185   gdp( 21, 00H,    018H, ALLVALID, 3, NULL,   12, 19) !stores & loads
186   gdp( 22, 00H,    038H, ALLVALID, 3, NULL,   13, 19) !double inc/dec
187   gdp( 23, 00H,    038H, ALLVALID, 3, NULL,   14, 19) !rotates
188   gdp( 24, 00H,    018H, ALLVALID, 3, NULL,   15, 19) !xthl, xchg, di, ei
189
190   gdp( 25, 00H,    0FFH, ALLVALID, 2, NULL, NULL,  9) ! io/mem data byte
191   gdp( 26, 20H,    007H, ALLVALID, 3, NULL,   17, 29) ! Status group
192   gdp( 27, 80H,    0F8H, ALLVALID, 5,    0,   18, 12) ! MISC status for
193                                                       ! mnemonic display.
194   ! The following Group Descriptors provide a count of
195   ! data bytes read by certain instruction groups.
196
197   gdp( 28, 00H,    03FH, ALLVALID, 6, NULL,   20,  0) ! MOV
198   gdp( 29, 00H,    018H, ALLVALID, 6, NULL,   21,  0) ! STAX...
199   gdp( 30, 00H,    018H, ALLVALID, 6, NULL,   22,  0) ! SHLD...
200   gdp( 31, 00H,    038H, ALLVALID, 6, NULL,   23,  0) ! MVI
201   gdp( 32, 00H,    018H, ALLVALID, 6, NULL,   24,  0) ! XTHL...
202   gdp( 33, 00H,    038H, ALLVALID, 6, NULL,   25,  0) ! POP & pals
203   gdp( 34, 00H,    000H, ALLVALID, 6, NULL,   26,  0) ! cond. returns
204   gdp( 35, 00H,    007H, ALLVALID, 6, NULL,   23,  0) ! arith & logical
205
206   !The following is a patch to fix the LXI command display.
207
208   gdp( 36, 02H,    0FFH, ALLVALID, 2, NULL, NULL, 27) ! immed hi byte
209   gdp( 37, 01H,    0FFH, ALLVALID, 2, NULL, NULL, 29) ! immed lo byte
210
211
212
213   ! Begining of Group Display Lists.
214
215   gdl( 0, 1,  'B', 'C', 'D', 'E', 'H', 'L', 'M', 'A')
216   gdl(16, 3, '7.5', '6.5', '5.5', 'INT', 'TRP')
217   gdl( 1, 3, 'ADD', 'ADC', 'SUB', 'SBB',
218              'ANA', 'XRA', 'ORA', 'CMP')
219   gdl( 2, 3, 'RNZ', 'RZ ', 'RNC', 'RC ',
220              'RPO', 'RPE', 'RP ', 'RM ')
```

```
221   gdl( 3, 8, 'POP  B  ', 'RET     ',
222              'POP  D  ', '********',
223              'POP  H  ', 'PCHL    ',
224              'POP  PSW', 'SPHL    ')
225   gdl( 4, 3, 'JNZ', 'JZ ', 'JNC', 'JC ', 'JPO', 'JPE', 'JP ', 'JM ')
226   gdl( 5, 3, 'OUT', 'IN ')
227   gdl( 6, 3, 'CNZ', 'CZ ', 'CNC', 'CC ',
228              'CPO', 'CPE', 'CP ', 'CM ')
229   gdl( 7, 8, 'PUSH B  ', 'PUSH D  ',
230              'PUSH H  ', 'PUSH PSW')
231
232   gdl( 8, 2, 'AD', 'AC', 'SU', 'SB', 'AN', 'XR', 'OR', 'CP')
233   gdl( 9, 3, 'NOP', '---', 'RIM', 'SIM')
234   gdl(10, 2, 'B ', 'D ', 'H ', 'SP')
235   gdl(11, 6, 'STAX B', 'LDAX B', 'STAX D', 'LDAX D')
236   gdl(12, 4, 'SHLD', 'LHLD', 'STA ', 'LDA ')
237   gdl(13, 7, 'INX  B ', 'DCX  B ',
238              'INX  D ', 'DCX  D ',
239              'INX  H ', 'DCX  H ',
240              'INX  SP', 'DCX  SP')
241   gdl(14, 3, 'RLC', 'RRC', 'RAL', 'RAR',
242              'DAA', 'CMA', 'STC', 'CMC')
243   gdl(15, 4, 'XTHL', 'XCHG', 'DI  ', 'EI  ')
244   gdl(17, 6, 'HALT  ', 'WRITE ', 'READ  ', 'FETCH ',
245              'HALT  ', 'OUTPUT', 'INPUT ', 'INTACK')
246   gdl(18, 1, '765IT') ! Interrupt indicators, mnemonic display.
247   gdl(19, 1, '01234567') ! RST instruction operands.
248
249   !  The following group display lists contain read sample counts.
250
251   gdl(20,  0, 0, 0, 0, 0, 0, 1, 0,     ! MOV B, x
252            0, 0, 0, 0, 0, 0, 1, 0,     ! MOV C, x
253            0, 0, 0, 0, 0, 0, 1, 0,     ! MOV D, x
254            0, 0, 0, 0, 0, 0, 1, 0,     ! MOV E, x
255            0, 0, 0, 0, 0, 0, 1, 0,     ! MOV H, x
256            0, 0, 0, 0, 0, 0, 1, 0,     ! MOV L, x
257            1, 1, 1, 1, 1, 1, 0, 1,     ! MOV M, x
258            0, 0, 0, 0, 0, 0, 1, 0)     ! MOV A, x
259
260   gdl(21,  0, 1, 0, 1)                 ! STAX...
261   gdl(22,  0, 2, 0, 1)                 ! SHLD...
262   gdl(23,  0, 0, 0, 0, 0, 0, 1, 0)     ! MVI
263   gdl(24,  2, 0, 0, 0)                 ! XTHL...
264   gdl(25,  2, 2, 2, 0, 2, 0, 2, 0)     ! POP & pals
265   gdl(26,  2)                          ! conditional returns
266
267
268   !The end.
```

```
*** WARNING At THR          - default value used
*** WARNING At VAR          - default value used
*** WARNING At FOL          - default value used
*** WARNING At VER          - default value used


Source file :            A:D8085U-01.DAT
Object file :            A:D8085U-01.EXE
List file   :            A:D8085U-01.LST
  268  Line(s) read
    0  Error(s) found
    4  Warning(s) issued
 2643  Bytes used
12385  Bytes remaining
```

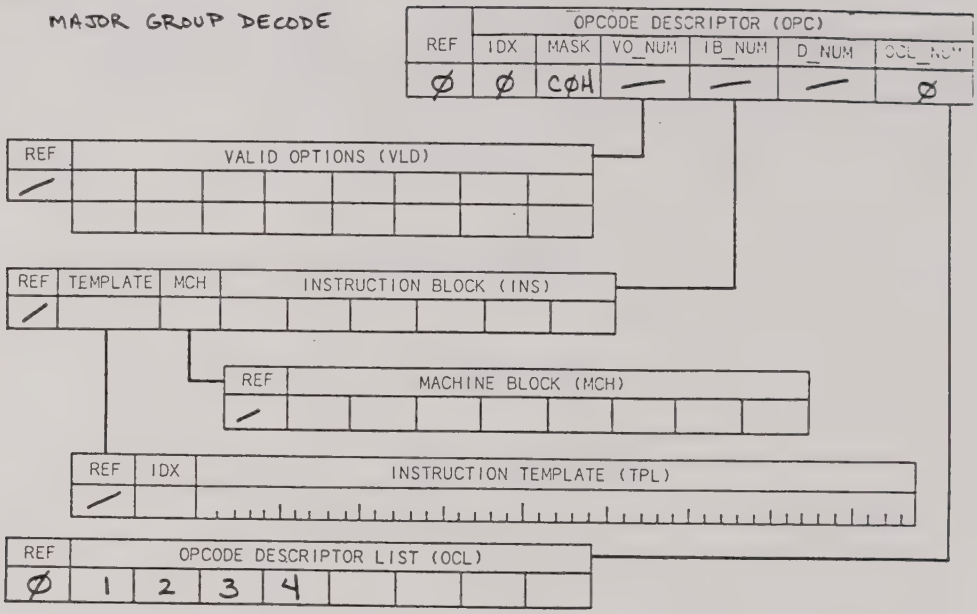
SAMPLE PREPROCESSOR WORKSHEETS, 8085 MICROPROCESSOR

Appendix C contains worksheets used to develop the UDD 8085 database listed in Appendix B. The first fifteen worksheets, pages C-2 through C-16, are OPC (Opcode Descriptor) Worksheets, and the last nineteen worksheets, pages C-17 through C-35, are GDP (Group Descriptor) Worksheets.

MICROPROCESSOR 8085 OPC WORKSHEET Page 1 of 15 Pages

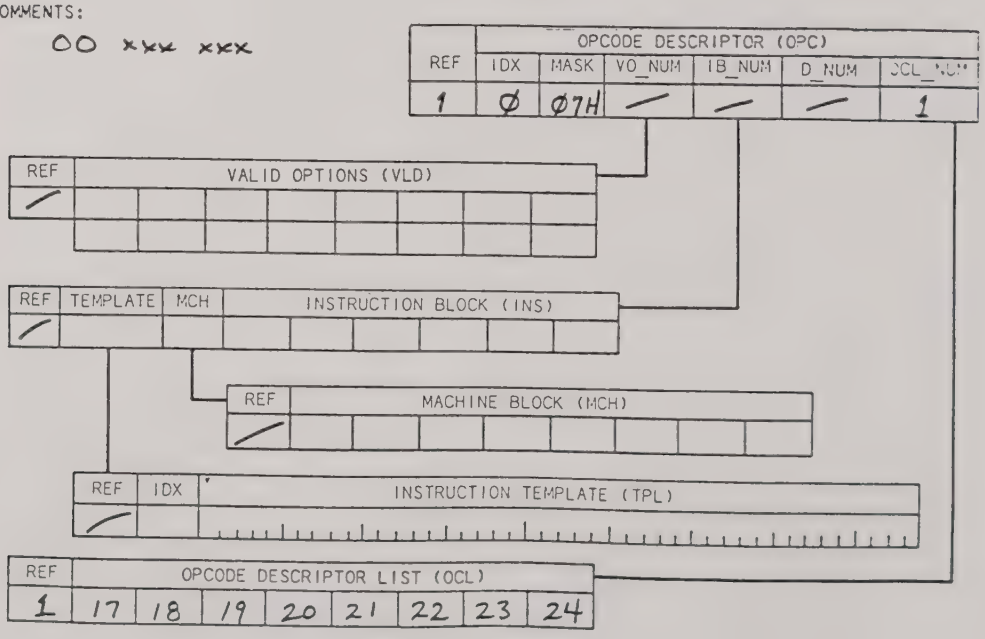
COMMENTS:

MAJOR GROUP DECODE



COMMENTS:

ØØ xxx xxx



COMMENTS:

01 xxx xxx
"HLT" (EXCEPTION)

REF	OPCODE DESCRIPTOR (OPC)					
	IDX	MASK	VO_NUM	IB_NUM	D_NUM	OCL_NUM
2	Ø	3FH	Ø	Ø	5	/

REF	VALID OPTIONS (VLD)								
Ø	36H								

REF	TEMPLATE	MCH	INSTRUCTION BLOCK (INS)						
Ø	Ø	1							

REF	MACHINE BLOCK (MCH)								
1	2	3	27						

REF	IDX	INSTRUCTION TEMPLATE (TPL)
Ø	19	HLT

REF	OPCODE DESCRIPTOR LIST (OCL)								
/									

COMMENTS:

10 x xx xxx
Arithmetic & Logical

REF	OPCODE DESCRIPTOR (OPC)					
	IDX	MASK	VO_NUM	IB_NUM	D_NUM	OCL_NUM
3	Ø	3FH	/	2	/	/

REF	VALID OPTIONS (VLD)								
/									

REF	TEMPLATE	MCH	INSTRUCTION BLOCK (INS)						
2	2	1	1	7	35				

REF	MACHINE BLOCK (MCH)								
/									

REF	IDX	INSTRUCTION TEMPLATE (TPL)
2	19	(empty)

REF	OPCODE DESCRIPTOR LIST (OCL)								
/									

MICROPROCESSOR 8085 OPC WORKSHEET Page 3 of 15 Pages

COMMENTS:

11 xxx xxx

REF	OPCODE DESCRIPTOR (OPC)					
	IDX	MASK	VO_NUM	IB_NUM	D_NUM	OCL_NUM
4	Ø	Ø7	/	/	/	2

REF	VALID OPTIONS (VLD)
/	

REF	TEMPLATE	MCH	INSTRUCTION BLOCK (INS)
/			

REF	MACHINE BLOCK (MCH)
/	

REF	IDX	INSTRUCTION TEMPLATE (TPL)
/		

REF	OPCODE DESCRIPTOR LIST (OCL)							
2	6	7	8	9	10	11	12	13

COMMENTS:

01 xxx xxx
"MOV" group

REF	OPCODE DESCRIPTOR (OPC)					
	IDX	MASK	VO_NUM	IB_NUM	D_NUM	OCL_NUM
5	Ø	3FH	/	1	/	/

REF	VALID OPTIONS (VLD)
/	

REF	TEMPLATE	MCH	INSTRUCTION BLOCK (INS)					
1	1	1	Ø	1	28			

REF	MACHINE BLOCK (MCH)
/	

REF	IDX	INSTRUCTION TEMPLATE (TPL)
1	19	MOV

REF	OPCODE DESCRIPTOR LIST (OCL)
/	

COMMENTS:

CONDITIONAL RETURNS

REF	OPCODE DESCRIPTOR (OPC)					
	IDX	MASK	VO_NUM	IB_NUM	D_NUM	OCL_NUM
6	Ø	38H	—	22	—	—

REF	VALID OPTIONS (VLD)
—	

REF	TEMPLATE	MCH	INSTRUCTION BLOCK (INS)
22	2	1	8 34

REF	MACHINE BLOCK (MCH)
—	

REF	IDX	INSTRUCTION TEMPLATE (TPL)
—		

REF	OPCODE DESCRIPTOR LIST (OCL)
—	

COMMENTS:

"POP" & FRIENDS

REF	OPCODE DESCRIPTOR (OPC)					
	IDX	MASK	VO_NUM	IB_NUM	D_NUM	OCL_NUM
7	0	38H	—	3	—	—

REF	VALID OPTIONS (VLD)
—	

REF	TEMPLATE	MCH	INSTRUCTION BLOCK (INS)
3	2	1	9 33

REF	MACHINE BLOCK (MCH)
—	

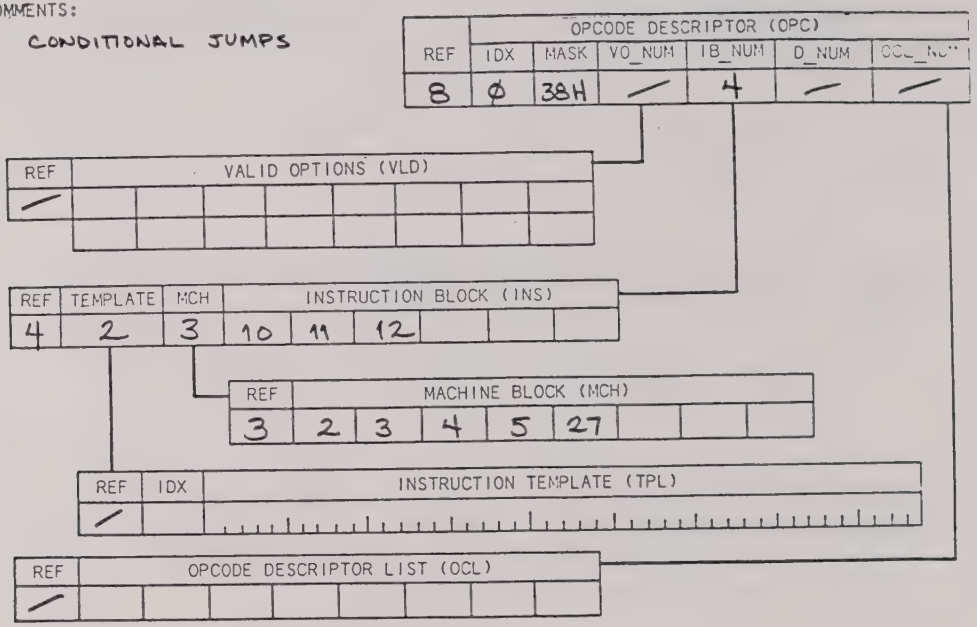
REF	IDX	INSTRUCTION TEMPLATE (TPL)
—		

REF	OPCODE DESCRIPTOR LIST (OCL)
—	

MICROPROCESSOR 8085 OPC WORKSHEET Page 5 of 15 Pages

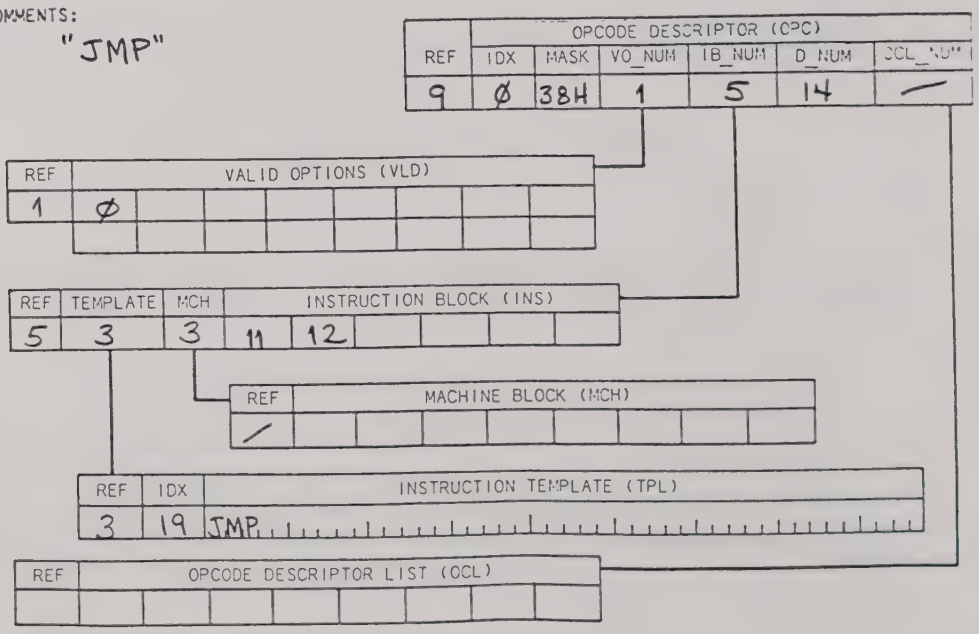
COMMENTS:

CONDITIONAL JUMPS



COMMENTS:

"JMP"



COMMENTS:

CONDITIONAL CALLS

REF	OPCODE DESCRIPTOR (OPC)					
	IDX	MASK	VO_NUM	IB_NUM	D_NUM	OCL_NUM
10	0	38H	/	7	/	/

REF	VALID OPTIONS (VLD)
/	

REF	TEMPLATE	MCH	INSTRUCTION BLOCK (INS)
7	2	3	14 11 12

REF	MACHINE BLOCK (MCH)
/	

REF	IDX	INSTRUCTION TEMPLATE (TPL)
/		

REF	OPCODE DESCRIPTOR LIST (OCL)
/	

COMMENTS:

"CALL"

REF	OPCODE DESCRIPTOR (OPC)					
	IDX	MASK	VO_NUM	IB_NUM	D_NUM	OCL_NUM
11	0	38H	3	8	16	/

REF	VALID OPTIONS (VLD)
3	1

REF	TEMPLATE	MCH	INSTRUCTION BLOCK (INS)
8	4	3	11 12

REF	MACHINE BLOCK (MCH)
/	

REF	IDX	INSTRUCTION TEMPLATE (TPL)
4	19	CALL

REF	OPCODE DESCRIPTOR LIST (OCL)
/	

COMMENTS:

IMMEDIATE ARITHMETIC & LOGICAL

REF	OPCODE DESCRIPTOR (OPC)					
	IDX	MASK	VO_NUM	IB_NUM	D_NUM	OCL_NUM
12	Ø	38H	—	1Ø	—	—

REF	VALID OPTIONS (VLD)							
—								

REF	TEMPLATE	MCH	INSTRUCTION BLOCK (INS)					
1Ø	5	2	12	16				

REF	MACHINE BLOCK (MCH)							
2	2	3	4	27				

REF	IDX	INSTRUCTION TEMPLATE (TPL)
5	21	I,

REF	OPCODE DESCRIPTOR LIST (OCL)							
—								

COMMENTS:

"RST x"

REF	OPCODE DESCRIPTOR (OPC)					
	IDX	MASK	VO_NUM	IB_NUM	D_NUM	OCL_NUM
13	Ø	38H	—	11	—	—

REF	VALID OPTIONS (VLD)							
—								

REF	TEMPLATE	MCH	INSTRUCTION BLOCK (INS)					
11	6	1	17					

REF	MACHINE BLOCK (MCH)								
—									

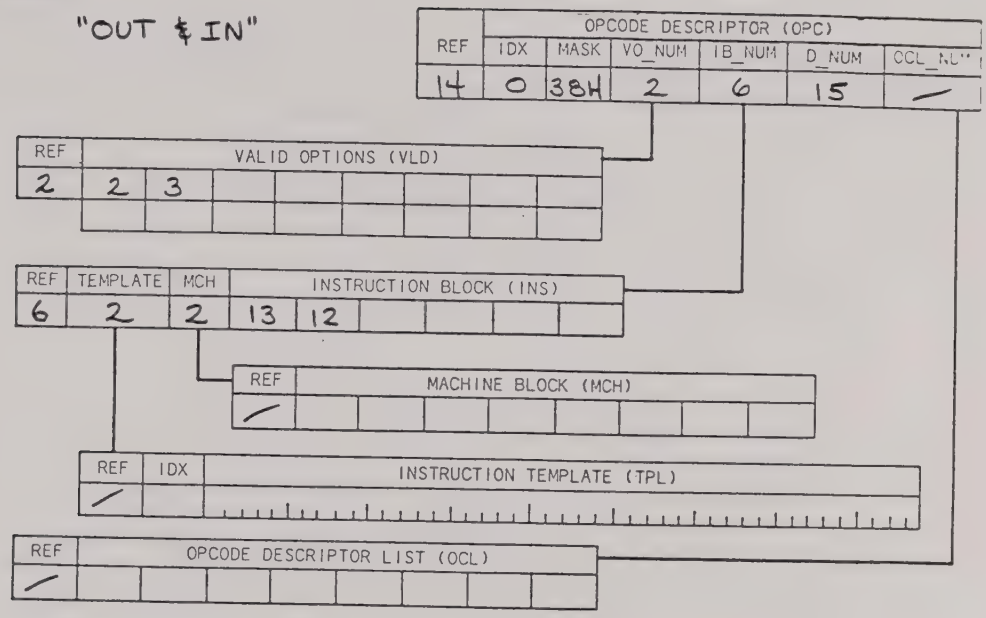
REF	IDX	INSTRUCTION TEMPLATE (TPL)
6	19	RST

REF	OPCODE DESCRIPTOR LIST (OCL)							
—								

MICROPROCESSOR 8085 OPC WORKSHEET Page 8 of 15 Pages

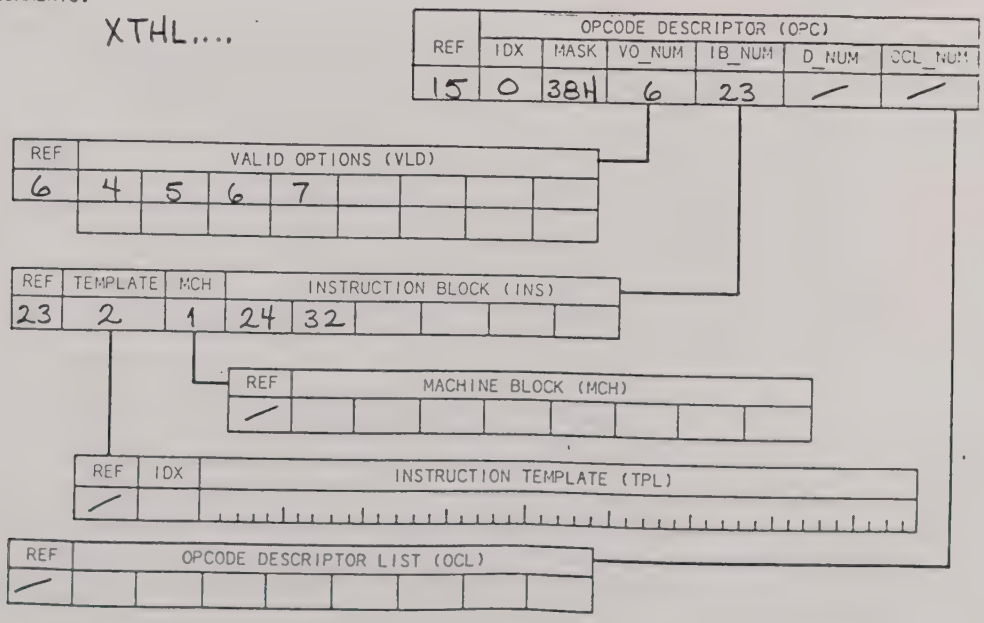
COMMENTS:

"OUT & IN"



COMMENTS:

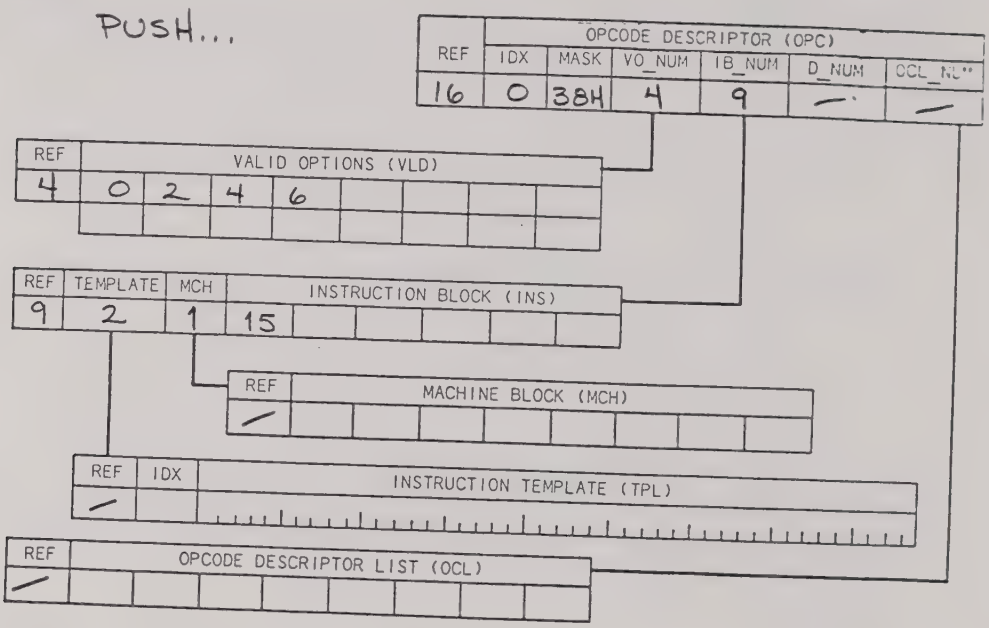
XTHL....



MICROPROCESSOR 8085 OPC WORKSHEET Page 9 of 15 Pages

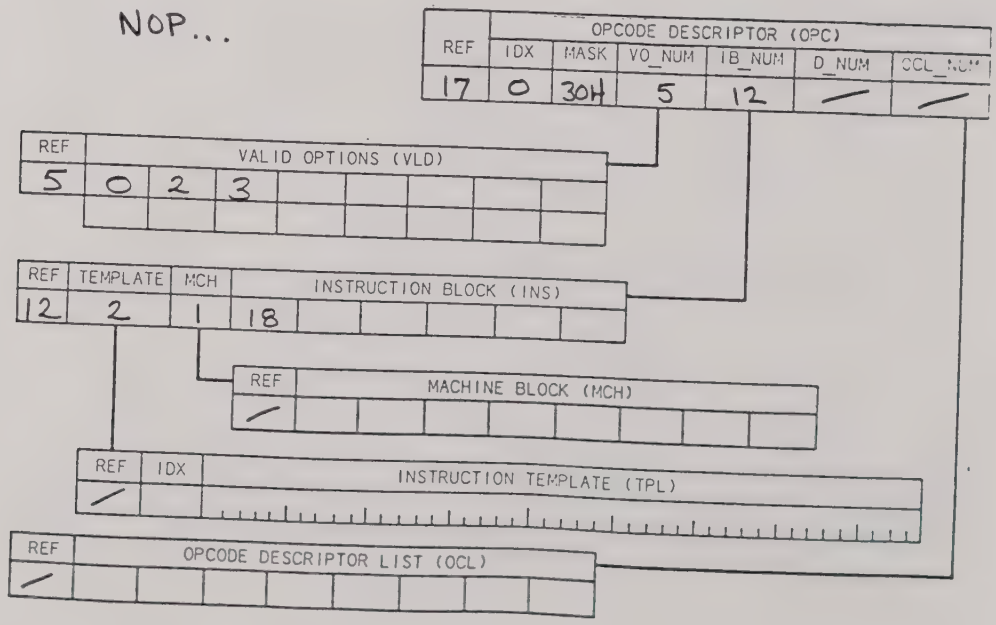
COMMENTS:

PUSH...



COMMENTS:

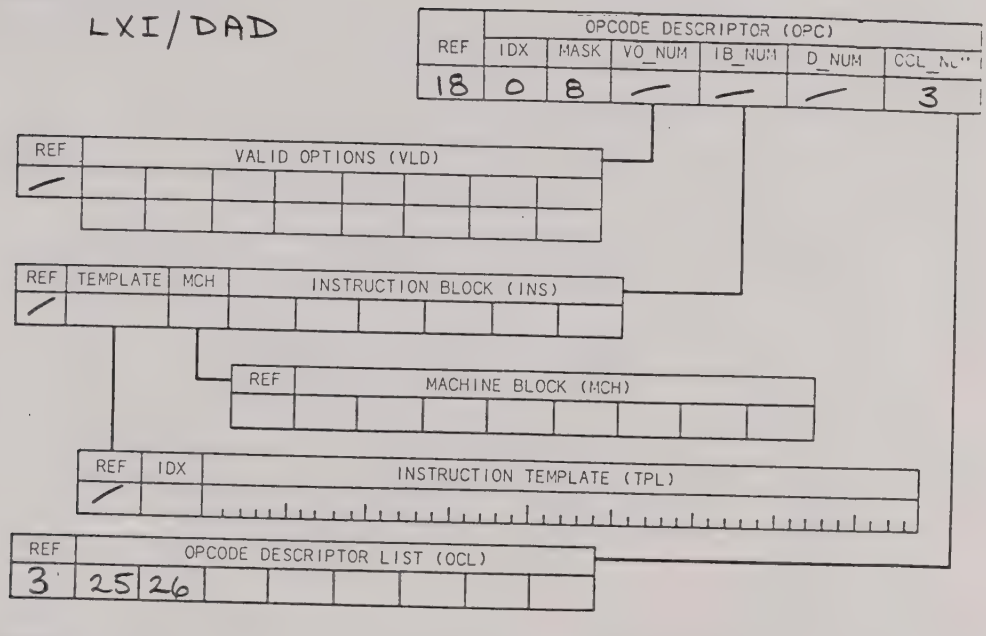
NOP...



MICROPROCESSOR 8085 OPC WORKSHEET

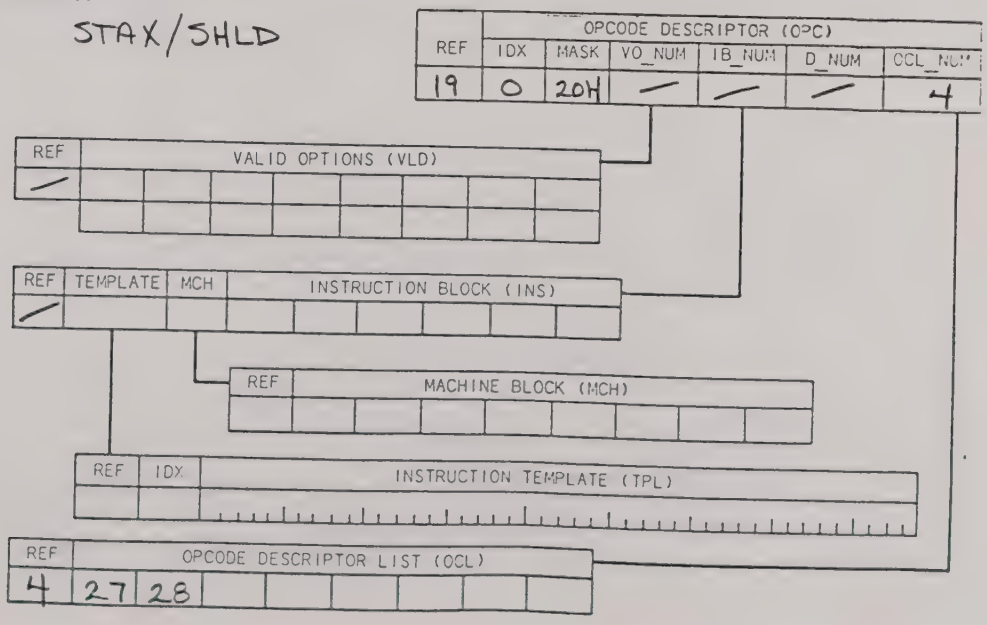
COMMENTS:

LXI/DAD



COMMENTS:

STAX/SHLD



COMMENTS:

INX/DCX

REF	OPCODE DESCRIPTOR (OPC)					
	IDX	MASK	VO_NUM	IB_NUM	D_NUM	OCL_NUM
20	0	38H	—	17	—	—

REF	VALID OPTIONS (VLD)
—	

REF	TEMPLATE	MCH	INSTRUCTION BLOCK (INS)
17	2	1	22

REF	MACHINE BLOCK (MCH)
—	

REF	IDX	INSTRUCTION TEMPLATE (TPL)
—		

REF	OPCODE DESCRIPTOR LIST (OCL)
—	

COMMENTS:

"INR"

REF	OPCODE DESCRIPTOR (OPC)					
	IDX	MASK	VO_NUM	IB_NUM	D_NUM	OCL_NUM
21	0	38H	—	18	—	—

REF	VALID OPTIONS (VLD)
—	

REF	TEMPLATE	MCH	INSTRUCTION BLOCK (INS)
18	8	1	0 31

REF	MACHINE BLOCK (MCH)
—	

REF	IDX	INSTRUCTION TEMPLATE (TPL)
8	19	INR

REF	OPCODE DESCRIPTOR LIST (OCL)
—	

COMMENTS:
"DCR"

REF	OPCODE DESCRIPTOR (OPC)					
	IDX	MASK	VO_NUM	IB_NUM	D_NUM	CCL_NUM
22	0	38H	—	19	—	—

REF	VALID OPTIONS (VLD)
—	

REF	TEMPLATE	MCH	INSTRUCTION BLOCK (INS)
19	9	1	0 31

REF	MACHINE BLOCK (MCH)
—	

REF	IDX	INSTRUCTION TEMPLATE (TPL)
9	19	DCR

REF	OPCODE DESCRIPTOR LIST (OCL)
—	

COMMENTS:
"MVI"

REF	OPCODE DESCRIPTOR (OPC)					
	IDX	MASK	VO_NUM	IB_NUM	D_NUM	CCL_NUM
23	0	38H	—	20	—	—

REF	VALID OPTIONS (VLD)
—	

REF	TEMPLATE	MCH	INSTRUCTION BLOCK (INS)
20	10	2	0 12 31

REF	MACHINE BLOCK (MCH)
—	

REF	IDX	INSTRUCTION TEMPLATE (TPL)
10	19	MVI

REF	OPCODE DESCRIPTOR LIST (OCL)
—	

COMMENTS:

ROTATES

REF	OPCODE DESCRIPTOR (OPC)					
	IDX	MASK	VO_NUM	IB_NUM	D_NUM	OCL_NUM
24	0	38H	—	21	—	—

REF	VALID OPTIONS (VLD)							
—								

REF	TEMPLATE	MCH	INSTRUCTION BLOCK (INS)					
21	2	1	23					

REF	MACHINE BLOCK (MCH)							
—								

REF	IDX	INSTRUCTION TEMPLATE (TPL)
—		

REF	OPCODE DESCRIPTOR LIST (OCL)							
—								

COMMENTS:

"LXI"

REF	OPCODE DESCRIPTOR (OPC)					
	IDX	MASK	VO_NUM	IB_NUM	D_NUM	OCL_NUM
25	0	30H	—	13	—	—

REF	VALID OPTIONS (VLD)							
—								

REF	TEMPLATE	MCH	INSTRUCTION BLOCK (INS)					
13	12	3	19	36	37			

REF	MACHINE BLOCK (MCH)							
—								

REF	IDX	INSTRUCTION TEMPLATE (TPL)
12	19	LXI

REF	OPCODE DESCRIPTOR LIST (OCL)							
—								

COMMENTS:

"DAD"

REF	OPCODE DESCRIPTOR (OPC)					
	IDX	MASK	VO_NUM	IB_NUM	D_NUM	OCL_NUM
26	0	30H	—	14	—	—

REF	VALID OPTIONS (VLD)
—	

REF	TEMPLATE	MCH	INSTRUCTION BLOCK (INS)
14	7	1	19

REF	MACHINE BLOCK (MCH)
—	

REF	IDX	INSTRUCTION TEMPLATE (TPL)
7	19	D,AD

REF	OPCODE DESCRIPTOR LIST (OCL)
—	

COMMENTS:

STAX ...

REF	OPCODE DESCRIPTOR (OPC)					
	IDX	MASK	VO_NUM	IB_NUM	D_NUM	OCL_NUM
27	0	18H	—	15	—	—

REF	VALID OPTIONS (VLD)
—	

REF	TEMPLATE	MCH	INSTRUCTION BLOCK (INS)	
15	2	1	20	29

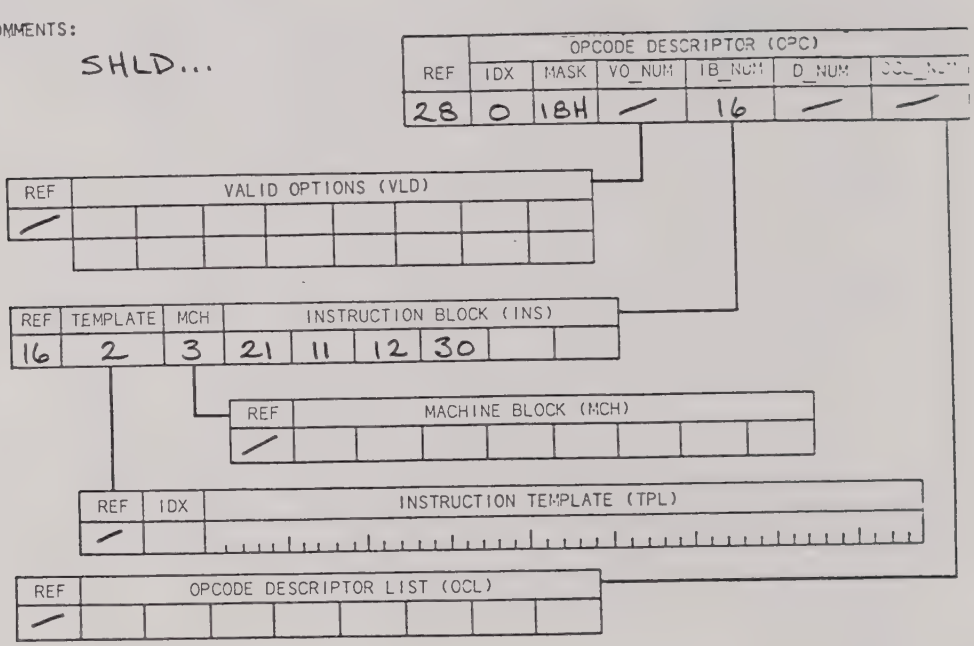
REF	MACHINE BLOCK (MCH)
—	

REF	IDX	INSTRUCTION TEMPLATE (TPL)
—		

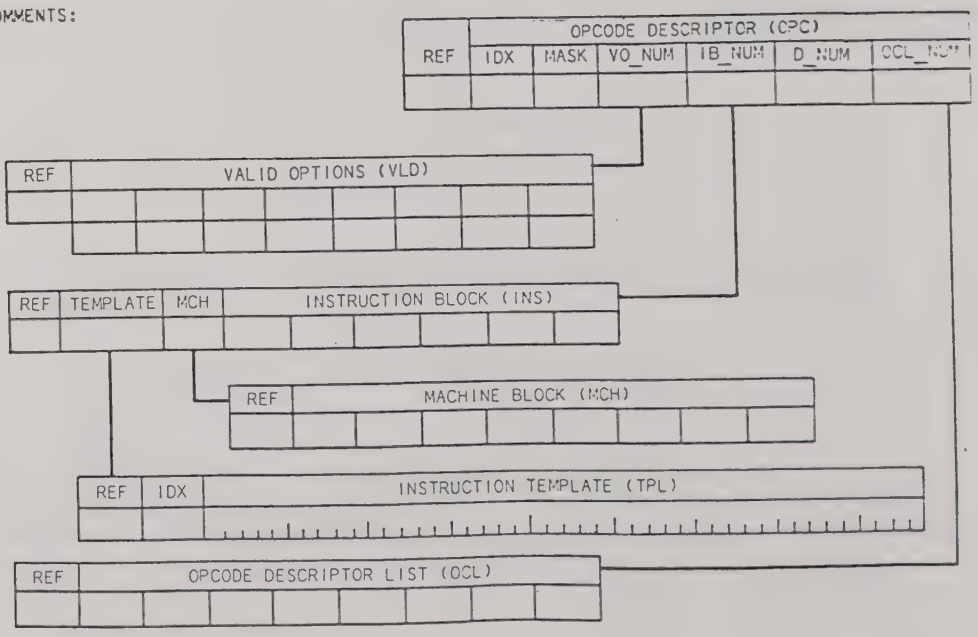
REF	OPCODE DESCRIPTOR LIST (OCL)
—	

COMMENTS:

SHLD...



COMMENTS:



MICROPROCESSOR 8085 GDP WORKSHEET Page 1 of 19 Pages

COMMENTS:

DESTINATION REGISTER

REF	GROUP DESCRIPTOR (GDP)						
	IDX	MASK	VO_NUM	DT	DT_PAR	GDL_NUM	OB_IDX
Ø	Ø	38H	—	3	—	Ø	24

REF	VALID OPTIONS (VLD)							
—								

REF	LEN	GROUP DESCRIPTOR LIST (GDL)	
Ø	1	B	C
		D	E
		H	L
		M	A

COMMENTS:

REF	GROUP DESCRIPTOR (GDP)						
	IDX	MASK	VO_NUM	DT	DT_PAR	GDL_NUM	OB_IDX
1	Ø	Ø7H	—	3	—	Ø	26

REF	VALID OPTIONS (VLD)							
—								

REF	LEN	GROUP DESCRIPTOR LIST (GDL)	
—			

COMMENTS:
MACHINE BLOCK
ADDRESS

REF	GROUP DESCRIPTOR (GDP)						
	IDX	MASK	VO_NUM	DT	DT_PAR	GDL_NUM	OB_IDX
2	40H	FFFF	/	2	/	/	0

REF	VALID OPTIONS (VLD)							
/								

REF	LEN	GROUP DESCRIPTOR LIST (GDL)
/		

COMMENTS:
MACHINE BLOCK
OBJECT BYTE 1

REF	GROUP DESCRIPTOR (GDP)						
	IDX	MASK	VO_NUM	DT	DT_PAR	GDL_NUM	OB_IDX
3	0	FF	/	2	/	/	5

REF	VALID OPTIONS (VLD)							
/								

REF	LEN	GROUP DESCRIPTOR LIST (GDL)
/		

MICROPROCESSOR 8085 GDP WORKSHEET Page 3 of 19 Pages

COMMENTS:
MACHINE BLOCK
OBJECT BYTE 2

REF	GROUP DESCRIPTOR (GDP)						
	IDX	MASK	VO_NUM	DT	DT_PAR	GDL_NUM	OB_IDX
4	1	FF	/	2	/	/	7

REF	VALID OPTIONS (VLD)							
/								

REF	LEN	GROUP DESCRIPTOR LIST (GDL)
/		

COMMENTS:
MACHINE BLOCK
OBJECT BYTE 3

REF	GROUP DESCRIPTOR (GDP)						
	IDX	MASK	VO_NUM	DT	DT_PAR	GDL_NUM	OB_IDX
5	2	FF	/	2	/	/	9

REF	VALID OPTIONS (VLD)							
/								

REF	LEN	GROUP DESCRIPTOR LIST (GDL)
/		

COMMENTS:
MISC GROUP DISPLAY

REF	GROUP DESCRIPTOR (GDP)						
	IDX	MASK	VO_NUM	DT	DT_PAR	GDL_NUM	OB_IDX
6	80H	F8H	—	5	0	16	12

REF	VALID OPTIONS (VLD)							
—								

REF	LEN	GROUP DESCRIPTOR LIST (GDL)	
16	3	7.5	6.5
		5.5	INT
		TRP	

COMMENTS:
ARITHMETIC & LOGICAL

REF	GROUP DESCRIPTOR (GDP)						
	IDX	MASK	VO_NUM	DT	DT_PAR	GDL_NUM	OB_IDX
7	0	38H	—	3	—	1	19

REF	VALID OPTIONS (VLD)							
—								

REF	LEN	GROUP DESCRIPTOR LIST (GDL)	
1	3	ADD	ADC
		SUB	SBB
		ANA	XRA
		ORA	CMP

COMMENTS:
CONDITIONAL RETURNS

REF	GROUP DESCRIPTOR (GDP)						
	IDX	MASK	VO_NUM	DT	DT_PAR	GDL_NUM	OB_IDX
8	0	38H	—	3	—	2	19

REF	VALID OPTIONS (VLD)							
—								

REF	LEN	GROUP DESCRIPTOR LIST (GDL)	
2	3	RNZ	RZ
		RNC	RC
		RPO	RPE
		RP	RM

COMMENTS:
POP & FRIENDS

REF	GROUP DESCRIPTOR (GDP)						
	IDX	MASK	VO_NUM	DT	DT_PAR	GDL_NUM	OB_IDX
9	0	38H	—	3	—	3	19

REF	VALID OPTIONS (VLD)							
—								

REF	LEN	GROUP DESCRIPTOR LIST (GDL)	
3	8	POP B	RET
		POP D	—
		POP H	PCHL
		POP PSW	SPHL

MICROPROCESSOR 8085 GDP WORKSHEET Page 6 of 19 Pages

COMMENTS:
CONDITIONAL JUMPS

REF	GROUP DESCRIPTOR (GDP)						
	IDX	MASK	VO_NUM	DT	DT_PAR	GDL_NUM	OB_IDX
10	0	38H	/	3	/	4	19

REF	VALID OPTIONS (VLD)								
/									

REF	LEN	GROUP DESCRIPTOR LIST (GDL)	
4	3	JNZ	JZ
		JNC	JC
		JPO	JPE
		JP	JM

COMMENTS:
INSTRUCTION BLOCK
HI BYTE ADDRESS

REF	GROUP DESCRIPTOR (GDP)						
	IDX	MASK	VO_NUM	DT	DT_PAR	GDL_NUM	OB_IDX
11	2	FF	/	2	/	/	24

REF	VALID OPTIONS (VLD)								
/									

REF	LEN	GROUP DESCRIPTOR LIST (GDL)	
/			

COMMENTS:
INSTRUCTION BLOCK
LO BYTE ADDRESS

REF	GROUP DESCRIPTOR (GDP)						
	IDX	MASK	VO_NUM	DT	DT_PAR	GDL_NUM	OB_IDX
12	1	FF	/	2	/	/	26

REF	VALID OPTIONS (VLD)							
/								

REF	LEN	GROUP DESCRIPTOR LIST (GDL)	
/			

COMMENTS:
IN & OUT

REF	GROUP DESCRIPTOR (GDP)						
	IDX	MASK	VO_NUM	DT	DT_PAR	GDL_NUM	OB_IDX
13	0	08H	/	3	/	5	19

REF	VALID OPTIONS (VLD)							
/								

REF	LEN	GROUP DESCRIPTOR LIST (GDL)	
5	3	OUT	IN

COMMENTS:
CONDITIONAL CALLS

REF	GROUP DESCRIPTOR (GDP)						
	IDX	MASK	VO_NUM	DT	DT_PAR	GDL_NUM	OB_IDX
14	0	38H	—	3	—	6	19

REF	VALID OPTIONS (VLD)							
—								

REF	LEN	GROUP DESCRIPTOR LIST (GDL)	
6	3	CNZ	CZ
		CNC	CC
		CPO	CPE
		CP	CM

COMMENTS:
DOUBLE REG. PUSHES

REF	GROUP DESCRIPTOR (GDP)						
	IDX	MASK	VO_NUM	DT	DT_PAR	GDL_NUM	OB_IDX
15	0	30H	—	3	—	7	19

REF	VALID OPTIONS (VLD)							
—								

REF	LEN	GROUP DESCRIPTOR LIST (GDL)	
7	8	PUSH B	PUSH D
		PUSH H	PUSH PSW

COMMENTS:

IMMEDIATE ARITHMETIC & LOGICAL

REF	GROUP DESCRIPTOR (GDP)						
	IDX	MASK	VO_NUM	DT	DT_PAR	GDL_NUM	OB_IDX
16	0	38H	/	3	/	8	19

REF	VALID OPTIONS (VLD)							
/								

REF	LEN	GROUP DESCRIPTOR LIST (GDL)	
8	2	AD	AC
		SU	SB
		AN	XR
		OR	CP

COMMENTS:

RST LEVEL

REF	GROUP DESCRIPTOR (GDP)						
	IDX	MASK	VO_NUM	DT	DT_PAR	GDL_NUM	OB_IDX
17	0	38H	/	3	/	19	24

REF	VALID OPTIONS (VLD)							
/								

REF	LEN	GROUP DESCRIPTOR LIST (GDL)	
19	1	01234567	

MICROPROCESSOR 8085 GDP WORKSHEET Page 10 of 19 Pages

COMMENTS:

MISCELLANEOUS
(NOP & PALS)

REF	GROUP DESCRIPTOR (GDP)						
	IDX	MASK	VO_NUM	DT	DT_PAR	GDL_NUM	OB_IDX
18	0	30H	—	3	—	9	19

REF	VALID OPTIONS (VLD)							
—								

REF	LEN	GROUP DESCRIPTOR LIST (GDL)	
9	3	NOP	—
		RIM	SIM

COMMENTS:

DOUBLE REG. NAMES

REF	GROUP DESCRIPTOR (GDP)						
	IDX	MASK	VO_NUM	DT	DT_PAR	GDL_NUM	OB_IDX
19	0	30H	—	3	—	10	24

REF	VALID OPTIONS (VLD)							
—								

REF	LEN	GROUP DESCRIPTOR LIST (GDL)	
10	2	B	D
		H	SP

COMMENTS:

INDIRECT MOVES

REF	GROUP DESCRIPTOR (GDP)						
	IDX	MASK	VO_NUM	DT	DT_PAR	GDL_NUM	OB_IDX
20	0	18H	—	3	—	11	19

REF	VALID OPTIONS (VLD)								
—									

REF	LEN	GROUP DESCRIPTOR LIST (GDL)	
11	6	STAX B	LDAX B
		STAX D	LDAX D

COMMENTS:

STORES & LOADS

REF	GROUP DESCRIPTOR (GDP)						
	IDX	MASK	VO_NUM	DT	DT_PAR	GDL_NUM	OB_IDX
21	0	18H	—	3	—	12	19

REF	VALID OPTIONS (VLD)								
—									

REF	LEN	GROUP DESCRIPTOR LIST (GDL)	
12	4	SHLD	LHLD
		STA	LDA

COMMENTS:

DOUBLE REG INC/DEC

REF	GROUP DESCRIPTOR (GDP)						
	IDX	MASK	VO_NUM	DT	DT_PAR	GDL_NUM	OB_IDX
22	0	38H	—	3	—	13	19

REF	VALID OPTIONS (VLD)								
—									

REF	LEN	GROUP DESCRIPTOR LIST (GDL)	
13	7	INX B	DCX B
		INX D	DCX D
		INX H	DCX H
		INX SP	DCX SP

COMMENTS:

ROTATES

REF	GROUP DESCRIPTOR (GDP)						
	IDX	MASK	VO_NUM	DT	DT_PAR	GDL_NUM	OB_IDX
23	0	38H	—	3	—	14	19

REF	VALID OPTIONS (VLD)								
—									

REF	LEN	GROUP DESCRIPTOR LIST (GDL)	
14	3	RLC	RRC
		RAL	RAR
		DAA	CMA
		STC	CMC

COMMENTS:
XTHL & FRIENDS

REF	GROUP DESCRIPTOR (GDP)						
	IDX	MASK	VO_NUM	DT	DT_PAR	GDL_NUM	OB_IDX
24	0	18H	/	3	/	15	19

REF	VALID OPTIONS (VLD)							
/								

REF	LEN	GROUP DESCRIPTOR LIST (GDL)	
15	4	XTHL	XCHG
		DI	EI

COMMENTS:
MACHINE BLOCK
IO/MEM DATA BYTE

REF	GROUP DESCRIPTOR (GDP)						
	IDX	MASK	VO_NUM	DT	DT_PAR	GDL_NUM	OB_IDX
25	0	FF	/	2	/	/	9

REF	VALID OPTIONS (VLD)							
/								

REF	LEN	GROUP DESCRIPTOR LIST (GDL)	
/			

COMMENTS:
OBJECT DISPLAY
STATUS GROUP

REF	GROUP DESCRIPTOR (GDP)						
	IDX	MASK	VO_NUM	DT	DT_PAR	GDL_NUM	OB_IDX
26	20H	07H	/	3	/	17	29

REF	VALID OPTIONS (VLD)							
/								

REF	LEN	GROUP DESCRIPTOR LIST (GDL)	
17	6	HALT	WRITE
		READ	FETCH
		HALT	OUTPUT
		INPUT	INTACK

COMMENTS:
MISC GROUP DISPLAY
(MNEMONIC MODE)

REF	GROUP DESCRIPTOR (GDP)						
	IDX	MASK	VO_NUM	DT	DT_PAR	GDL_NUM	OB_IDX
27	80H	F8H	/	5	0	18	12

REF	VALID OPTIONS (VLD)							
/								

REF	LEN	GROUP DESCRIPTOR LIST (GDL)	
18	1	765IT	

COMMENTS:

MOV INSTRUCTION
DATA BYTES READ

REF	GROUP DESCRIPTOR (GDP)						
	IDX	MASK	VO_NUM	DT	DT_PAR	GDL_NUM	CB_IDX
28	0	3FH	/	6	/	20	0

REF	VALID OPTIONS (VLD)							
/								

REF	LEN	GROUP DESCRIPTOR LIST (GDL)	
20	/	00000010	(B)
		00000010	(C)
		00000010	(D)
		00000010	(E)
		00000010	(H)
		00000010	(L)
		11111101	(M)
		00000010	(A)

COMMENTS:

STAX ETC.
DATA BYTES READ

REF	GROUP DESCRIPTOR (GDP)						
	IDX	MASK	VO_NUM	DT	DT_PAR	GDL_NUM	CB_IDX
29	0	18H	/	6	/	21	0

REF	VALID OPTIONS (VLD)							
/								

REF	LEN	GROUP DESCRIPTOR LIST (GDL)	
21	/	0,1,0,1	

COMMENTS:
SHLD etc
DATA BYTES READ

REF	GROUP DESCRIPTOR (GDP)						
	IDX	MASK	VO_NUM	DT	DT_PAR	GDL_NUM	OB_IDX
30	0	18H	—	6	—	22	0

REF	VALID OPTIONS (VLD)							
—								

REF	LEN	GROUP DESCRIPTOR LIST (GDL)	
22	—	0,2,0,1	

COMMENTS:
MVI
DATA BYTES READ

REF	GROUP DESCRIPTOR (GDP)						
	IDX	MASK	VO_NUM	DT	DT_PAR	GDL_NUM	OB_IDX
31	0	38H	—	6	—	23	0

REF	VALID OPTIONS (VLD)							
—								

REF	LEN	GROUP DESCRIPTOR LIST (GDL)	
23	—	0,0,0,0,0,0,1,0	

COMMENTS:
XTHL ETC
DATA BYTES READ

REF	GROUP DESCRIPTOR (GDP)						
	IDX	MASK	VO_NUM	DT	DT_PAR	GDL_NUM	CB_IDX
32	0	18H	/	6	/	24	0

REF	VALID OPTIONS (VLD)								
/									

REF	LEN	GROUP DESCRIPTOR LIST (GDL)	
24	/	2,0,0,0	

COMMENTS:
POP & PALS
DATA BYTES READ

REF	GROUP DESCRIPTOR (GDP)						
	IDX	MASK	VO_NUM	DT	DT_PAR	GDL_NUM	CB_IDX
33	0	38H	/	6	/	25	0

REF	VALID OPTIONS (VLD)								
/									

REF	LEN	GROUP DESCRIPTOR LIST (GDL)	
25	/	2,2,2,0,2,0,2,0	

COMMENTS:
CONDITIONAL RETURNS
DATA BYTES READ

REF	GROUP DESCRIPTOR (GDP)						
	IDX	MASK	VO_NUM	DT	DT_PAR	GDL_NUM	OB_IDX
34	0	0	—	6	—	26	0

REF	VALID OPTIONS (VLD)							
—								

REF	LEN	GROUP DESCRIPTOR LIST (GDL)
26	—	2

COMMENTS:
ARITH. & LOGICAL
DATA BYTES READ

REF	GROUP DESCRIPTOR (GDP)						
	IDX	MASK	VO_NUM	DT	DT_PAR	GDL_NUM	OB_IDX
35	0	07H	—	6	—	23	0

REF	VALID OPTIONS (VLD)							
—								

REF	LEN	GROUP DESCRIPTOR LIST (GDL)
—		

COMMENTS:

LXI
HI BYTE DATA

REF	GROUP DESCRIPTOR (GDP)						
	IDX	MASK	VO_NUM	DT	DT_PAR	GDL_NUM	OB_IDX
36	2	FF	/	2	/	/	26

REF	VALID OPTIONS (VLD)							
/								

REF	LEN	GROUP DESCRIPTOR LIST (GDL)	
/			

COMMENTS:

LXI
LO BYTE DATA

REF	GROUP DESCRIPTOR (GDP)						
	IDX	MASK	VO_NUM	DT	DT_PAR	GDL_NUM	OB_IDX
37	1	FF	/	2	/	/	28

REF	VALID OPTIONS (VLD)							
/								

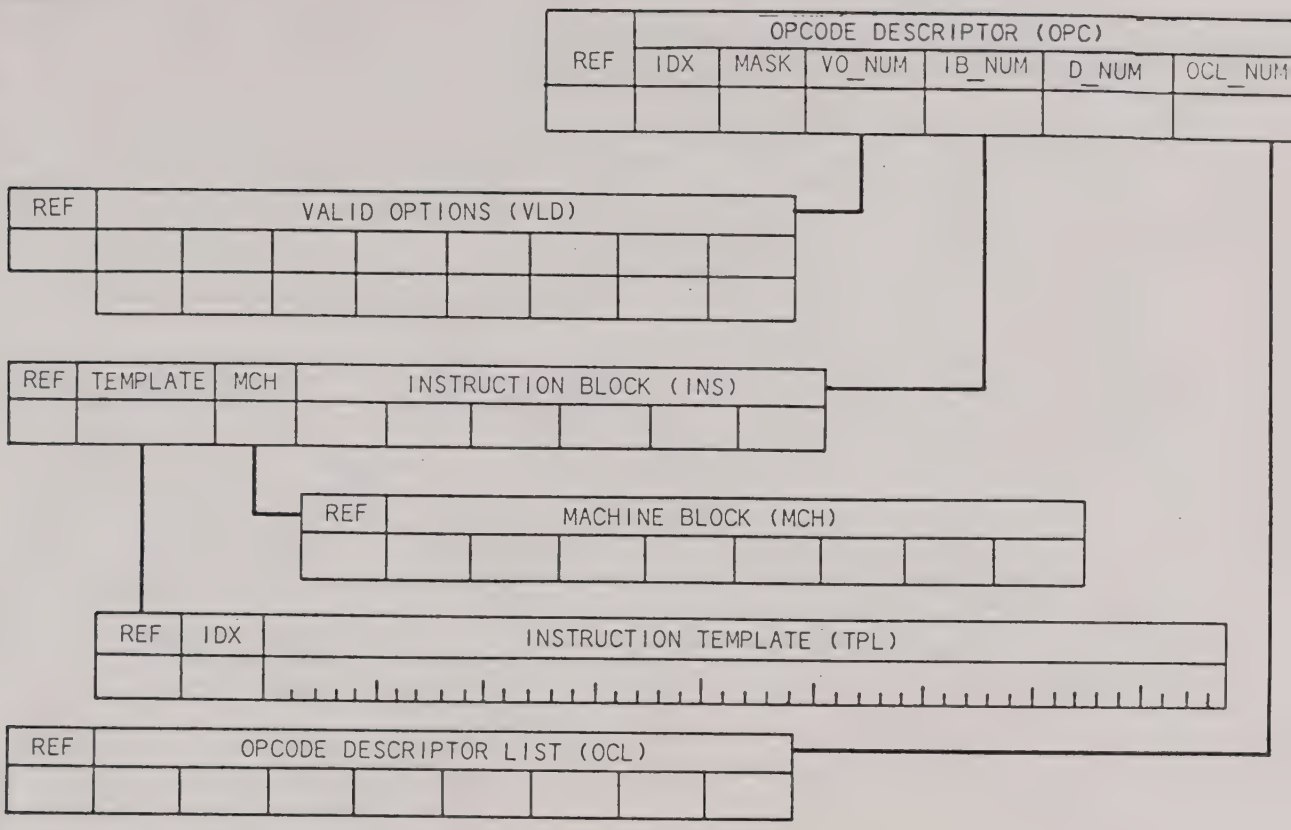
REF	LEN	GROUP DESCRIPTOR LIST (GDL)	
/			

PREPROCESSOR WORKSHEETS

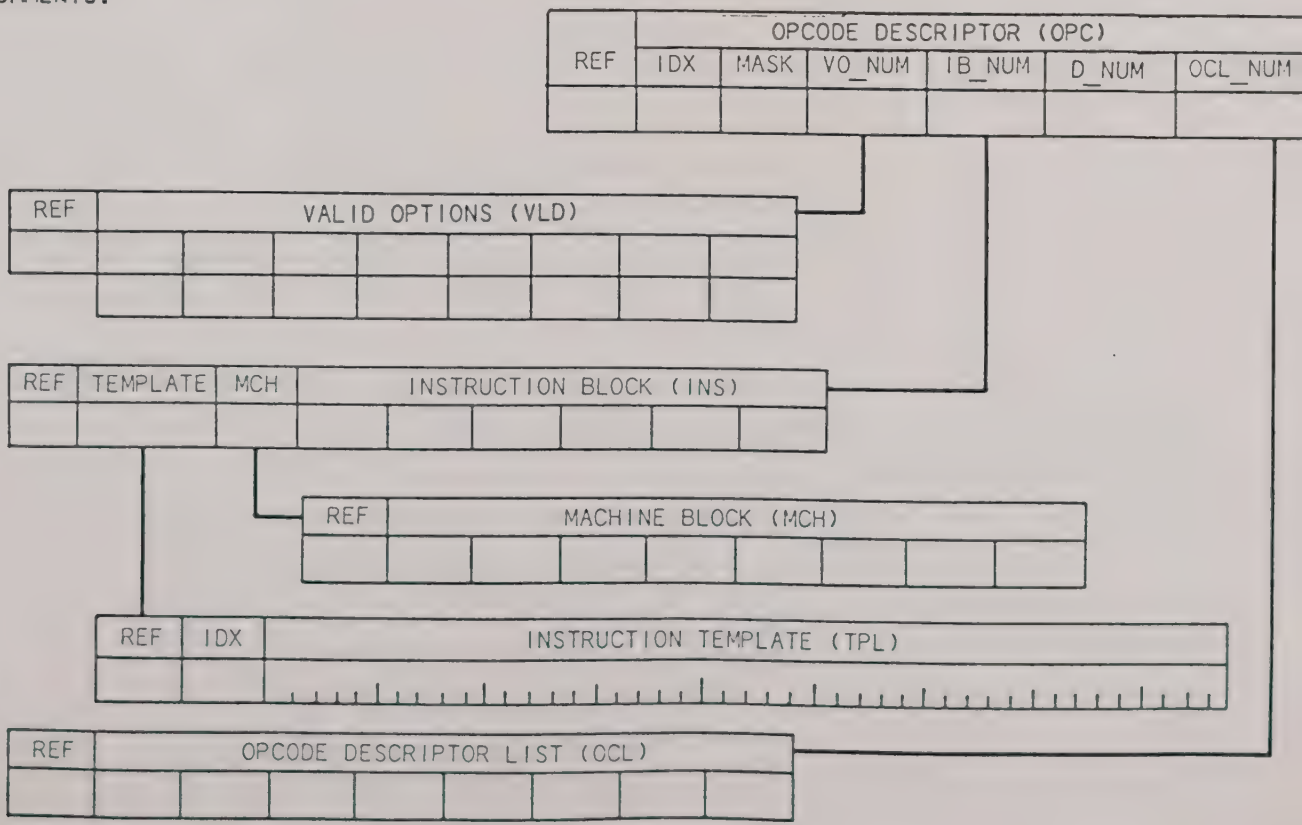
Make copies of the worksheets on the following pages and use them to keep track of the UDD decode statements and associated arrays.

MICROPROCESSOR _____ OPC WORKSHEET Page ___ of ___ Pages

COMMENTS:



COMMENTS:



MICROPROCESSOR _____ GDP WORKSHEET Page ___ of ___ Pages

COMMENTS:

REF	GROUP DESCRIPTOR (GDP)						
	IDX	MASK	VO_NUM	DT	DT_PAR	GDL_NUM	OB_IDX

REF	VALID OPTIONS (VLD)							

REF	LEN	GROUP DESCRIPTOR LIST (GDL)	

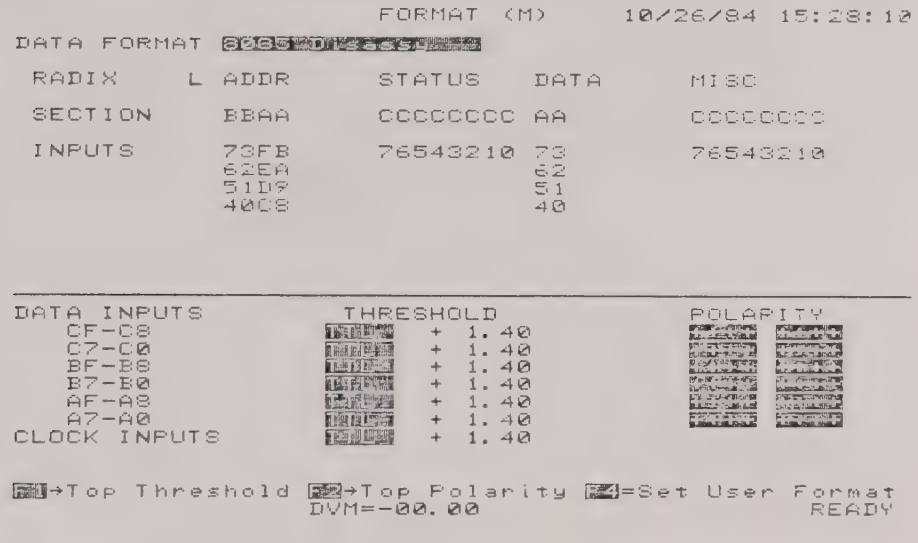
COMMENTS:

REF	GROUP DESCRIPTOR (GDP)						
	IDX	MASK	VO_NUM	DT	DT_PAR	GDL_NUM	OB_IDX

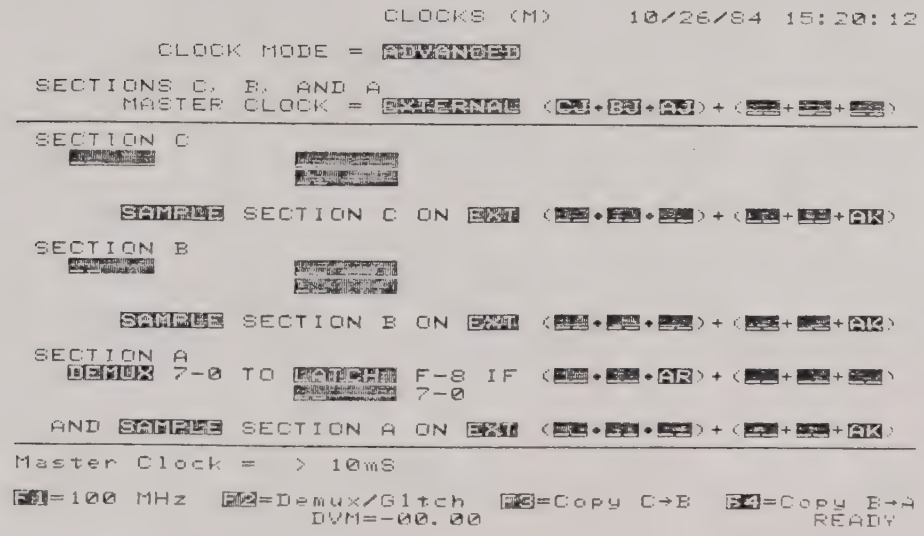
REF	VALID OPTIONS (VLD)							

REF	LEN	GROUP DESCRIPTOR LIST (GDL)	

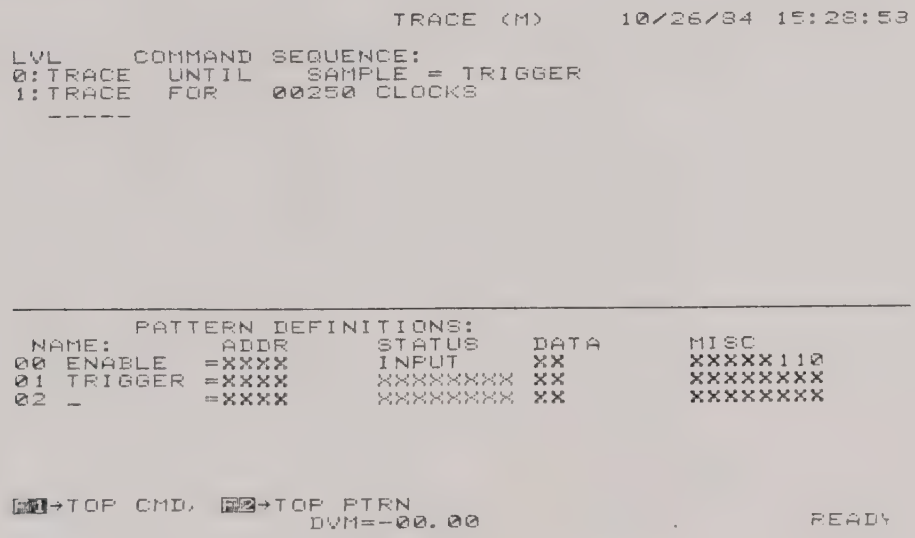
SAMPLE DISASSEMBLER FORMAT SCREEN



SAMPLE DISASSEMBLER CLOCK SCREEN


CLOCKS (M) 10/26/84 15:20:12
CLOCK MODE = ADVANCED
SECTIONS C, B, AND A
MASTER CLOCK = EXTERNAL (CJ • BJ • AJ) + (+ +)
SECTION C
SAMPLE SECTION C ON EXT (• •) + (+ + AK)
SECTION B
SAMPLE SECTION B ON EXT (• •) + (+ + AK)
SECTION A
DEMUX 7-0 TO F-8 IF (• • AR) + (+ +)
7-0
AND SAMPLE SECTION A ON EXT (• •) + (+ + AK)
Master Clock = > 10mS
F1=100 MHz F2=Demux/Glitch F3=Copy C→B F4=Copy B→A
DVM=-00.00
READY

SAMPLE DISASSEMBLER TRACE CONTROL SCREEN



SAMPLE DISASSEMBLER DATA DISPLAY SCREEN

```
                    8085 Disassy     (A) 10/25/84 10:53:50

     S=         XXXX XXXXXXXX          XX XXXXXXXX              X
  FRAME  ADDR  OBJECT  STATUS  MNEMONIC
C  32    0259  D600              SUI     00                     1
   34    025B  C6FF              ADI     FF                     1
   36    025D  9F                SBB     A                      1
   37    025E  326BC0            STA   C06B                     1
   40    C06B        FF                        WRITE            1
   41    0261  C9                RET                            1
   42    C0FA        34                        READ             1
   43    C0FB        02                        READ             1
   44    0234  3A6BC0            LDA   C06B                     1
   47    C06B        FF                        READ             1
   48    0237  1F                RAR                            1
   49    0238  DA3E02            JC    023E                     1
   52    023E  216AC0            LXI   H ,C06A                  1
   55    0241  34                INR   M                        1
   56    C06A        E2                        READ             1
   57    C06A        E3                        WRITE            1
   58    0242  C22302            JNZ   0223                     1
   61    0223  3EFF              MVI   A,FF                     1
   63    0225  216AC0            LXI   H ,C06A                  1
   66    0228  BE                CMP     M                      1
   67    C06A        E3                        READ             1

  CONTROL=032    REF=006            R-C=-  26
F1→PG UP F2→PG DOWN F3→DSPLY OBJ F4→DSPLY MNEMONICS
CLOCK=EXTERNAL    DVM=-00.00                          READY
```

ERROR AND WARNING MESSAGES

Appendix I lists the error and warning messages pertaining to the UDD-105 with a brief description of the conditions causing the error, where applicable.

- Missing left parenthesis -- Error --

  Parentheses must occur in pairs.

- Missing <CR> -- Error --

  A control-word statement should be followed by a <CR>.

- Missing Keyword -- Error --

  When a statement begins with a left parenthesis, a Keyword should be included.

- Missing right parenthesis -- Error --

  Parentheses must occur in pairs.

- Illegal statement -- Error --

  A statement should not begin with a single quote, right parenthesis, or a comma. ') ,

- Illegal "$", not a control statement -- Error --

  There should be no "$" in a statement unless the $ is contained in a string surrounded by single quotation marks. A dollar sign introduces a control statement and should be located at the beginning of a line only.

- Missing term -- Error --

  Pairs of parentheses must contain a term.

- Not a legal number -- Error --

  At least one character is not appropriate for the required radix. For example, 012B is not a legal number, because the "2" is not a member of the binary-number character set.

- Not a legal character: 0, 1, or X -- Error --

  The TCS specifier array requires 0, 1, or X characters only.

- Missing single quote -- Error --

  Single quotation marks must occur in pairs.

- Illegal syntax after right parenthesis -- Error --

  Only a <CR>, space, or "!" can follow a right parenthesis on a line.

- Incorrect number of terms -- Error --

  The number of defined terms is either too large or too small.

- Term is not correct type -- Error --

  This is a string_type field. Numerical input cannot be entered in this type of field.

- Term out of range -- Error --

  The value of term is too large or too small.

- Illegal Control Word -- Error --

- Not a legal Keyword -- Error --

- Not a legal term -- Error --

- Duplicated Keyword -- Error --

- length not equal -- Error --

  VTC array lengths are not equal.

- insufficient terms -- Error --

  The number of terms in CHK is not consistent with the number of terms defined by MSK.

- term not defined -- Error --

- no default value -- Error --

- PF1 & PFX must coexist -- Error --

  In CHK structure, PF1 and PFX must exist at the same time.

- String size out of range, truncated -- Warning --

  Only the first n characters of a string are displayed in a string designated as having length, n.

- Illegal NOOBJECT control ignored -- Warning --

  A NOOBJECT control occurs at the beginning of a source code only.

- Illegal NOPRINT control ignored -- Warning --

  A NOPRINT control occurs at the beginning of a source code only.

- Line too long, truncated -- Warning --

  The maximum number of characters for each line is 128.

- break in sequence -- Warning --

  There is a break in the reference numbers of two or more GDP, OPC, [ETC.] specifiers are not in sequence. [These may be out or sequence as long as there is not a break in the reference numbers -1, 2, 4, 5, 6 is wrong; 1, 3, 2, 6, 4, 5 is ok.]

- - default value used -- Warning --

- IB_NUM & OLC_NUM conflict -- Warning --

  IB_NUM and OCL_NUM cannot exist at the same time. If they did, OCL_NUM would be ignored.

- D_NUM is ignored -- Warning --

  If VO_NUM in the OC structure is ALL VALID (OFFFFH) then if the user enters a D_NUM valid, it will be ignored by the UDD.

RS-232 PORT SPECIFICATIONS

The UDD utilizes the auxiliary device connector on the rear panel of the K205 Logic Analzyzer as an RS-232 port. Any serial device conforming to the RS-232 specifications and meeting the needs of the UDD, terminal or printer, can be connected to this port.

Parameters for the RS-232 port are set up using the K205 I/O display screen. The I/O display screen is addressed when in normal operation (not in UDDGEN or UUTIL) by pressing the I/O key followed by numeric-key 2. Alterable parameters and their respective options are listed as follows:

1. Baud Rate:

   - 110
   - 150
   - 300
   - 600
   - 1200
   - 1800
   - 2400
   - 4800
   - 9600

2. Word Length:

   - 7 bits
   - 8 bits

3. Stop Bits:

   - 1 stop bit
   - 2 stop bits

4. Parity:

   - None
   - Even parity
   - Odd parity

5. Protocol:

   The UDD always uses XON/XOFF protocol, regardless of the setting on the K205 Setup screen. The UDD receiver also accepts the RTS/CTS protocol, but it is not recommended.

It is not necessary to select RS-232 as the active I/O port in the lower portion of the K205 Setup screen.

The K205 Logic Analyzer auxiliary device port is configured as a Data Terminal-Equipment (DTE) port with the following pin connections utilized:

1 - Chassis ground
2 - Transmit data
3 - Receive data
4 - RTS
5 - CTS

6 - DSR
7 - Signal ground
8 - No connection
9 - No connection
10 - No connection

11 - No connection
12 - No connection
13 - No connection
14 - No connection
15 - No connection

16 - No connection
17 - No connection
18 - No connection
19 - No connection
20 - DTR

21 - No connection
22 - No connection
23 - No connection
24 - No connection
25 - No connection

Connect the K205 to an auxiliary device with a null modem. These connections can be made through a break-out box or through cross connecting two RS232-C connectors. The null-modem connection, illustrated in Figure J-1, was used with the FX-80, Televideo 912B, and Televideo 920C terminals.

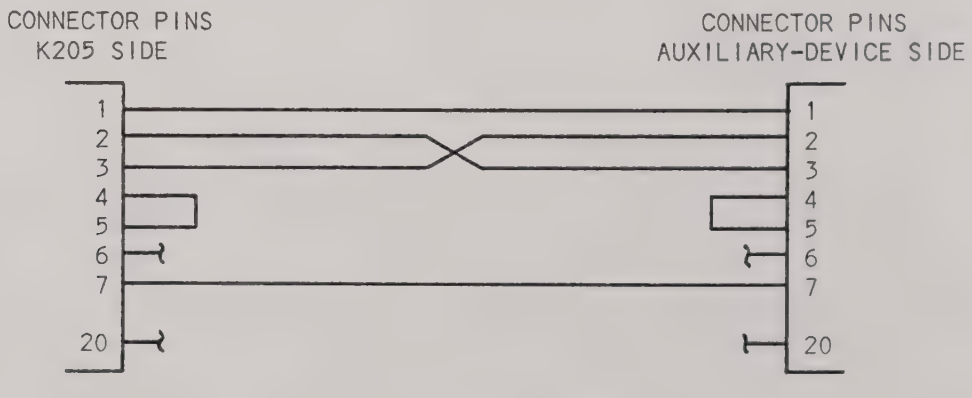


Figure J-1. Null Modem

ABBREVIATIONS

This appendix lists abbreviations and definitions used in this manual.

CHK	-	Status check
D_NUM	-	Default Opcode reference number used if the field value does not match an element of the VALID_OPTIONS array
DMD	-	Display mode
DRD	-	Display-read
DT	-	Display type
DT_PAR	-	Display-type parameter
GD_NUM	-	Reference number of GDP
GDP	-	Group descriptor
GP_LIST	-	Group list
GP_NUM	-	Reference number of GP_LIST
IB_NUM	-	Reference number of the Instruction Block for the opcode group
IDX	-	Index into the IN_DAT array, specifies the address group used as a source of displayed data
INS	-	Instruction block
INSTR_BLOCK	-	Used to build disassembly display line
M_NUM	-	MCH reference number in the INS specifier
MASK	-	Defines the size of a field
MCH	-	Machine block
MSK	-	Status mask
NODATA	-	Trace_align code representing a K105-D sample location that does not contain trace data (contains flush data)
OB_IDX	-	Output-buffer index
OCL_NUM	-	Reference number of the OC_LIST array containing list of Opcode reference numbers for each member of the current Opcode group

OPC	-	Opcode
PF1	-	First processor instruction word
PFX	-	Subsequent processor words
PILL	-	--- to designate an illegal status
PRD	-	Processor-read
PWR	-	Processor-write
REF	-	Reference designator
STAT_MSK	-	Defines used bits of TGP (status)
TAC	-	Trace Alignment Code
TCS	-	Defines the names and values of the status selection available on the (K105) Trace Control Screen
TGP	-	TRACE GROUP
T_NUM	-	Reference number of template specifier
TPL	-	Template
VLD	-	Valid_option array specifier
VO_NUM	-	Reference number of the Value_Options (VLD)
VTC	-	Valid trace codes

GOULD Design & Test Systems Division

The leader in technology, performance and innovation

NO POSTAGE NECESSARY IF MAILED IN THE UNITED STATES

BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 1173 SANTA CLARA, CA

POSTAGE WILL BE PAID BY ADDRESSEE

Gould, Inc.,
Design & Test Systems Division

4650 Old Ironsides Drive
Santa Clara, California 95054-9990

NO POSTAGE NECESSARY IF MAILED IN THE UNITED STATES

BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 1173 SANTA CLARA, CA

POSTAGE WILL BE PAID BY ADDRESSEE

Gould, Inc.,
Design & Test Systems Division

4650 Old Ironsides Drive
Santa Clara, California 95054-9990

Gould, Inc.

The leader in technology, performance and innovation

Reader Comments

The manual's completeness, accuracy, organization, usability, and reliability: _____

_____

Did you find errors in this manual: _____ How can this manual be improved: _____

_____

_____

Additional comments: _____

_____

Name _____ Department _____ M/S _____

Company: _____ Manual or Part No. _____

Address _____

City _____ State _____ Zip _____ Phone _____

Warranty/Registration

Please return this card to Gould, Inc. within five days

Did the packaging of this equipment exhibit any outward signs of physical damage ..... YES ○ ○ NO

Did this equipment arrive intact, without missing parts, loose parts or cable damage? ..... YES ○ ○ NO

Did the equipment operate on power-up? ..... YES ○ ○ NO

Did you attain adequate system performance? ..... YES ○ ○ NO

Were any electrical adjustments required? ..... YES ○ ○ NO

If you required assistance, was a local Gould, Inc. representative contacted? ..... YES ○ ○ NO

Comments _____

User Name _____ Title _____ Department _____ M/S _____

Company _____ Phone _____

Address _____ City _____ State _____ Zip _____

Model or Description _____ Serial or Part No. _____ Data Received _____